RT-25: Requirements Management for Net-Centric Enterprises

Phase I Report SERC-2011-TR-017

Douglas A. Bodner - Georgia Institute of Technology
Nenad Medvidovic - University of Southern California
William B. Rouse - Georgia Institute of Technology
Barry W. Boehm - University of Southern California
Richard A. DeMillo - Georgia Institute of Technology
George Edwards - University of Southern California
Daniyal Khan - Georgia Institute of Technology
Ivo Krka - University of Southern California
Jo Ann Lane - University of Southern California
Aditya Pradhan - Georgia Institute of Technology

Contract Number: H98230-08-D-0171 DO1, TTO2, RT25

Report No. SERC-2011-TR-017
April 28, 2011

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
28 APR 2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
RT-25: Requirements Management for Net-Centric Enterprises	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Stevens Institute of Technology,Systems Engineering Research Center (SERC),1 Castle Point on Hudson,Hoboken,NJ,07030**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
SERC is sponsored by the Department of Defense.

14. ABSTRACT

Net-centric enterprises increasingly are found in government and industry contexts. In this research, a net-centric enterprise consists of a number of semi-autonomous organizations that collaborate within the context of a federated structure. Such collaborations may be temporary and of known duration, temporary and of unknown duration, or permanent and known to be permanent. When such semi-autonomous organizations collaborate, they typically have information technology needs to support their collaboration. In the information technology (IT) domain, such needs are called requirements. From a business or organizational perspective, these needs are called capabilities or functions. In designing and developing IT systems to support high-level capabilities, capabilities are decomposed to functions and then to requirements. From requirements, software architectures are derived and then implemented. The fundamental problem is how to manage the process of proceeding from capabilities to systems, i.e., requirements management in the net-centric enterprise. The preceding simple linear process description is useful, but inadequate to address the complexity of the net-centric enterprise. This complexity manifests itself in the following forms ? the need to join existing IT systems belonging to the organizations involved in the collaboration to support the desired capabilities, the perhaps unknown durations of such collaborations, the presence of legacy systems, and the evolving needs and missions of the various organizations. This research uses case study analysis of business mergers and other types of IT integrations to aid in the specification of a methodology to address the requirements management problem. This report provides the results of a Phase 1 effort of this research, including case study analysis methodology specifications and recommendations for future research.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**66**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

This page intentionally left blank

ABSTRACT

Net-centric enterprises increasingly are found in government and industry contexts. In this research, a net-centric enterprise consists of a number of semi-autonomous organizations that collaborate within the context of a federated structure. Such collaborations may be temporary and of known duration, temporary and of unknown duration, or permanent and known to be permanent.

When such semi-autonomous organizations collaborate, they typically have information technology needs to support their collaboration. In the information technology (IT) domain, such needs are called requirements. From a business or organizational perspective, these needs are called capabilities or functions. In designing and developing IT systems to support high-level capabilities, capabilities are decomposed to functions and then to requirements. From requirements, software architectures are derived and then implemented. The fundamental problem is how to manage the process of proceeding from capabilities to systems, i.e., requirements management in the net-centric enterprise.

The preceding simple linear process description is useful, but inadequate to address the complexity of the net-centric enterprise. This complexity manifests itself in the following forms – the need to join existing IT systems belonging to the organizations involved in the collaboration to support the desired capabilities, the perhaps unknown durations of such collaborations, the presence of legacy systems, and the evolving needs and missions of the various organizations. This research uses case study analysis of business mergers and other types of IT integrations to aid in the specification of a methodology to address the requirements management problem. This report provides the results of a Phase 1 effort of this research, including case study analysis, methodology specifications and recommendations for future research.

This page intentionally left blank

TABLE OF CONTENTS

FIGURES AND TABLES

1 SUMMARY

This report details the findings of Phase 1 of a research effort studying requirements management in net-centric enterprises during the first six months. In this research, a net-centric enterprise consists of a number of semi-autonomous organizations that collaborate within the context of a federated structure. Such collaborations may be temporary and of known duration, temporary and of unknown duration, or permanent and known to be permanent. When such organizations collaborate, they have IT needs to support the collaboration. Since the organizations have pre-existing IT systems, the collaboration needs are often posed as a system integration or merger.

This report uses case study analysis of business mergers, system-of-systems and other types of system integrations to identify key success factors, problem areas and user needs. These case studies include the merger between Hewlett-Packard and Compaq, a DoD multi-platform system-of-systems, a regional area crisis management system-of-systems, health care IT integration to support sharing of patient records among multiple hospital providers/systems, a back-office IT integration for an internet service provider, and the failed FBI Virtual Case File System.

Existing methods, processes and tools that address various aspects of the requirements management problem are identified and described. Drawing on this framework, a methodology for requirements management in the net-centric context is specified, and places in the framework are identified where the various MPTs would serve a useful purpose.

The remainder of this report is organized as follows. Section 2 introduces the problem, discusses the research objectives and approach and provides definitions of technical terms. Section 3 details the state-of-the-art relative to requirements management in net-centric enterprises. Section 4 introduces a number of case studies that are used in the research and then provides a framework for assessing the case studies relative to a structured decision process to aid in system integration/merging in a net-centric context. Section 5 then identifies and describes MPTs relevant to the problem domain and proposes a methodology for the problem that incorporates the MPTs. Finally, Section 6 concludes and provides future research directions.

2 INTRODUCTION

2.1 PROBLEM STATEMENT

This research addresses requirements management in the net-centric enterprise. In this context, system integrations and mergers, often of a temporary or unspecified duration, are used to support multi-organizational collaboration. Requirements management then involves identifying, reconciling, documenting, analyzing and prioritizing capabilities, functions and requirements as capabilities are decomposed into requirements and then mapped to architectures. Requirements management should support the traceability of progress on the extent to which capabilities are being realized and should also support the evolution of new capabilities as the net-centric enterprise evolves to support new missions and collaborations.

This research specifically seeks to specify a methodology and associated MPTs to enable effective requirements management in a net-centric context.

2.2 OBJECTIVES

The overall objectives of this research are the following.

- Enable "requirements management" throughout integration lifecycle
  - Requirements definition and reconciliation
  - Traceability
  - Architecture specification
  - Balance between automation and decision support
- Address
  - Organizational differences
  - Selection-from-alternatives vs. design
  - Ambiguity and robustness

2.3 RESEARCH APPROACH

This research incorporates two principal perspectives – that of enterprise and business process systems engineering and that of information technology systems engineering. These two perspectives inform the nature of the problem, namely the decomposition of capabilities to requirements to architectures and the decision prioritization process whereby this is accomplished.

Using these two perspectives, case studies are analyzed within the context of a framework that is simultaneously evolved for understanding important considerations in addressing the problem. The case studies and framework are used to specify a methodology for addressing the problem.

2.4 DEFINITIONS

The following terms are used in this report as defined here.

- Architecture – A variety of definitions for architecture exist. One relevant definition for this research is that an architecture is "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" (ANSI/IEEE 2006). This is not meant to be exclusive of other, similar definitions maintained by other standards organizations.
- Capability – Consistent with DoD usage, a capability is the "ability to achieve a desired effect under specified standards and conditions through combinations of ways and means to perform a set of tasks" (CJCS 2007). More specifically, capability is a high-level business imperative that typically is decomposed into subordinate functions that together achieve the capability.
- Function – A function is an intermediate concept between a capability and a requirement. In a large, complex enterprise, there may be many levels of functions as capabilities are decomposed into functions, thence to requirements.
- Integration – A system integration occurs when two or more systems are joined by developing interfaces between them. Each system, more or less, retains its identity. An integration can be temporary.
- Interoperability – Interoperability is the property of a system whereby the system's interfaces are specifically designed to work with other systems with limited or no interface modifications. Inherent in this definition is that the interfaces are well-understood enough to accommodate working with other systems from an implementation perspective.
- Interoperation – Interoperation occurs when two or more systems are joined together using existing interoperability features of the systems. Interoperation is typically temporary.
- Merger – A system merger occurs when two or more systems are joined by taking the best parts of each and combining them to form a new system. Mergers are typically permanent.
- MPT – Methods, processes and tools are used to solve specific problems in the systems engineering domain.
- Net-centric enterprise – A net-centric enterprise engages a number of semi-autonomous organizations under the umbrella of a federated structure. These organizations have independent but related missions and often collaborate.

- Requirement – In IT systems, a requirement is a particular, well-defined and well-scoped need that must be satisfied by a system. A requirement typically has a stakeholder or stakeholders who advocate for its continued inclusion in the system design and development. In this research, a requirement is generally considered to be at a lower level (i.e., closer to the software design and development process) than a capability or function.
- Requirements definition – Requirements definition, in the traditional software engineering sense, refers to the process of determining requirements for a software system. Here, it also encompasses the determination of high-level capabilities.
- Requirements management – Requirements management is the process whereby capability intents are identified, decomposed into functions, then into requirements for system design, development and evolution. Conflicts and dependencies between capabilities, functions and requirements must be identified and tracked, with conflicts being resolved. Requirements (including capabilities and functions) must be documented, analyzed, and prioritized. Additionally, requirements management includes the traceability of progress toward meeting higher-level capabilities and functions and lower-level requirements are met. As the system's mission changes, new capabilities, functions and requirements may be added.
- Requirements reconciliation – Requirements reconciliation is a method/process by which different stakeholders in a system are brought together to determine their requirements (or more generally capabilities or functions) for the system, to identify any conflicts, and then negotiate a mutually agreeable solution based on prioritization.
- System-of-systems (SoS) – A system-of-systems is a large-scale, complex system composed of a collection of heterogeneous, independent components that themselves are considered systems. An SoS may be directed, in that it is designed, built and managed for a particular purpose (with the constituent systems normally under control of the overall SoS management), or it may be acknowledged, in that it is centrally managed with the constituent systems retaining their individual autonomy, and with any changes negotiated by the individual systems and the overall SoS management (DoD 2008).

3 STATE OF THE ART

Requirements management can be approached from two perspectives. First, is the IT perspective, whereby requirements are tracked though the software development process to ensure that the end product does, in fact, meet the requirements specified. The other perspective addresses business processes, whereby enterprise capabilities are decomposed into business processes that are designed to meet the enterprise capabilities. In particular, this section addresses requirements management in a net-centric enterprise, where mergers and integrations are of importance.

Mehta and Hirschheim (2004) present a framework for studying IT integration within the context of mergers and acquisitions. Specifically, they focus on appearances to external constituencies (e.g., Wall Street), power differentials between the organizations involved, and the nature of the business-IT strategic alignment in the merged organization. These are useful considerations that may help shape the requirements management problem.

A key issue is shown in Figure 1. An executive decision-maker is concerned with design, development and implementation of a capability. This entails decomposing the capability into requirements that can be used to guide software development. This decision-maker would like to know the progress towards realization of the capability. This implies the property of traceability, i.e., that somehow, the process on realizing requirements in the development process can be traced back to progress of realizing the capability. This is complicated, of course, in an environment consisting of mergers and integrations.

Existing tools address the traditional software system requirements management problem, which allows traceability of the meeting of requirements by developed and implemented software. However, what remains relatively unaddressed is traceability upwards in the enterprise to allow key decision-makers to know the process on a capability. It should be noted that this not only encompasses multiple organizational levels and multiple stakeholders (i.e., due to the merger/integration situation), but also potentially ambiguity and capture of rationale.

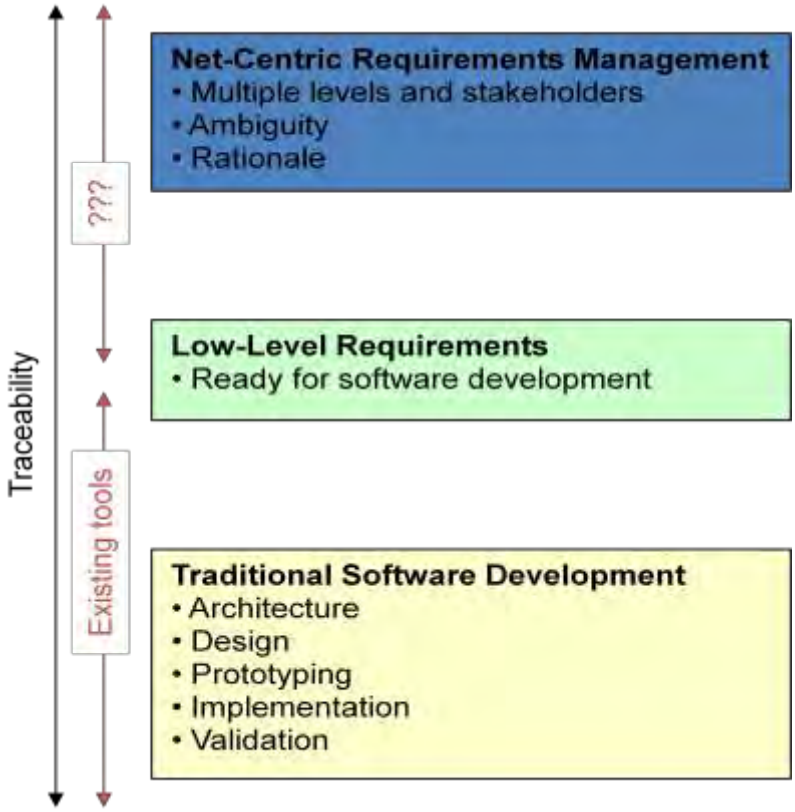

Figure 1: Traceability of requirements in a net-centric context

From a business process perspective, a variety of tools are available for specifying and modeling process-oriented software systems. These include Business Process Modeling Notation (BPMN) (White and Miers 2008) and Business Process Execution Language (BPEL) (Sarang, Juric et al. 2006). Such languages are useful in that they have a graphical component, similar to UML or SysML, and they map to simulation tools that can be used to assess process/system performance (Laguna and Marklund 2004). Strictly speaking, they are not tools for requirements management, although they can be used as decision aids for designing process-oriented IT systems and SOA systems, and they generally support hierarchical modeling.

Software requirements are traditionally captured using natural language statements accompanied with more formal models that include Use-Case diagrams, Unified Modeling Language (UML) diagrams (e.g., sequence diagrams, class diagrams), or Petri-Nets. While these general purpose models have proven useful in certain contexts, they lack the ability to appropriately capture the system intent. For the purpose of capturing the intentional requirements of software systems, researchers have proposed goal-oriented requirements engineering (GORE) practices (van Lamsweerde 2001).

In GORE, the central notion for capturing requirements is that of system goals, which are hierarchically decomposed using AND/OR relations. The requirements

decomposition facilitates traceability of higher-level requirements to lower-level ones, while also allowing a designer to capture variability or multiple candidate choices for realizing a goal. The system goals can be strong goals (e.g., a safety requirement) or soft goals (e.g., a desired latency level).

Over the period of the last 15 years, the GORE research has focused on ways of formalizing goals, detecting and resolving obstacles, and providing semi-automated reasoning about the realizability of system goals. Furthermore, several patters for decomposing a higher-level goal into lower-level goals have been proposed. In the context of net-centric enterprises, where each net-centric node has its own set of requirements, GORE practices can help to identify overlapping goals, missing goals, and conflicting goals, and in certain situations even help reconcile them (van Lamsweerde and Letier 2001). Additionally, current practices do not deal with the possibility that some parts of the goal hierarchy (e.g., goal corresponding to another net-centric node) may not be accessible.

Note that GORE primarily focuses on lower-level software system's goals, and does not go as far as modeling system business capabilities. Some of our previous work has addressed requirements reconciliation among multiple stakeholders, but not in a hierarchical or a dynamic net-centric enterprise with capability decomposition (Boehm, Grunbacher et al. 2001). More recent work has focused on decomposing capabilities to functions of constituent systems in a system-of-systems (SoS) (Lane and Bohn 2010). Furthermore, the recent Department of Defense's Systems Engineering Guide for Systems of Systems captures the best practices for engineering complex SoS (DoD 2008). A part of our effort will be to utilize and enhance these existing methods for applications in a dynamic net-centric enterprise where many decisions need to be made at a short time-scale and with differing durability. We cover the details of these approaches in Sections 5.1.1, 5.1.4, and 5.1.5, respectively.

To reiterate, the envisioned traceability would start with high-level system capabilities and map over requirements to the level of an IT system's architectural components. While the traceability between capabilities and requirements has been a largely unexplored problem, even the traceability between requirements and architectures is limited in the current state-of-the-art and state-of-the-practice. To bridge the gap between the system requirements and its architecture, we have proposed a methodology for mapping them via intermediate models, where the mapping is performed incrementally (Grünbacher, Egyed et al. 2004). Further details on this approach, which we plan to utilize in our future research, are described in Section 5.1.6.

At the level of IT systems, the research area of software architectures has identified the appropriate abstractions and codified the most important characteristics and concepts related to a system's architecture (Taylor, Medvidovic et al. 2009). Of particular importance for a net-centric enterprise is the codification of architectural styles, such as client-server, peer-to-peer, and publish-subscribe, which capture the most important

high-level interactions between system components. In a net-centric enterprise, each net-centric node may follow a particular architectural style, depending on the IT systems requirements. Hence, when an overarching capability needs to involve multiple systems, the disparities between the individually utilized styles may arise. The characterization of such architectural inconsistencies and identification of specifically compatible styles is lacking in the existing literature although it would help the process of integrating multiple IT systems (Land and Crnkovic 2011). This is one of the directions we intend to pursue in our future work with the goal of facilitating seamless integration of multiple independent IT systems in a net-centric enterprise.

4 CONCEPTS AND CASE STUDY SUMMARIES

Fundamental to this research is a process whereby case study analysis and methodology specification/enhancement are iteratively performed. This section presents the case studies used for this phase of the research, with the purpose of describing their relative successes and lack of success with respect to requirements engineering. Critical success factors and user need priorities are highlighted.

4.1 CASE STUDY SUMMARIES

Six case studies are described. These range from private industry (e.g., corporate mergers), to public sector (e.g., Department of Defense and local/state emergency response), to public-private regulated industries (e.g., health care).

It should be noted that several of the case studies have classified or proprietary aspects. Thus, not all features can be described in detail, in some cases including system identification.

4.1.1 HP-COMPAQ MERGER

On September 3, 2001, Hewlett-Packard (HP) announced that it had reached a deal with Compaq whereby the two companies would merge. HP had previously divested Agilent Technologies. Compaq had recently acquired Digital Equipment Corporation (DEC), and that merger had not been successful (Baldwin and Lane 2003). The HP-Compaq merger involved two similar organizations, both of which were large providers of computers and servers. In addition, HP had an existing business line in printers and an emerging IT services organization. Thus, the goal was cost consolidation in preparation for an expected upswing in demand.

However, the merger faced two approval hurdles – approval from U.S and European government regulatory agencies and approval of the shareholders. Approval of the shareholders was problematic, as an HP board member and son of a co-founder led an increasingly bitter and public proxy fight against the merger (Burgelman and Meza 2004). The proxy fight continued until March of 2002, when the merger received a very narrow victory. Beforehand, regulatory agencies in the U.S. and Europe had given their approval to the merger. The "first day" merger date was set for May 7, 2002.

The merger has been documented extensively from a business perspective (Baldwin and Lane 2003; Bell DeTienne and Hoopes 2004; Mark and Mitchell 2004; Palepu and Barnett 2004; Perlow and Kind 2004; Beer, Khurana et al. 2005). Clearly, the merger had implications for the IT systems of both companies. From analyzing the various case

study reports produced on the merger, the implications on IT systems from desired organizational capabilities can be detailed. This is shown in Table 1, for the pre-merger HP and Compaq companies and for the newly merged company. Of course, these capabilities evolved along a timeline, shown in Figure 2.

Company	Organizational Capability	Implications for IT Systems
HP	Divestment of Agilent	Separation and streamlining
HP	Front-back structure	Reorganization
HP	Integrated procurement consolidation	Legacy system consolidation, version management
Compaq	DEC and Tandem integration	Consolidation and streamlining
Compaq	Increased inventory turns	More responsive and transparent
HP-Compaq	Economies of scale, distribution and integration	Consolidation and rationalization. 'One Company'
HP-Compaq	'Near integration' of customer facing	Integrated internally but familiar externally.
HP-Compaq	'Adopt and Go', 'Launch and Learn', 'Fast Start'	Ability to copy
HP-Compaq	Vertical Horizontal orientation and Horizontal focus.	Reorganization to reflect new processes and capabilities
HP-Compaq	Leadership framework	Transparency, flexibility, effectiveness,
HP-Compaq	Greater SMB focus	Internet sales, horizontal integration
HP-Compaq	'Adaptive Enterprise'	Flexibility, re-organization.

Table 1: HP-Compaq capabilities and IT system implications

Prior to the merger, HP and Compaq engaged in extensive planning (Perlow and Kind 2004). They were constrained by legal considerations (i.e., anti-trust violations) pending approval of the merger. They set up a small integration office designated to meet in a "clean room." That is, the integration team (i.e., "clean team") had no contact with the rest of the operating businesses. Initially, they paired the corresponding HP and Compaq colleagues. As planning proceeded, the team grew larger and developed a technical roadmap for the merger, including IT systems. The "clean team" operated under a number of principles, chief among them Adopt-and-Go and Launch-and-Learn (Burgelman and Meza 2004).

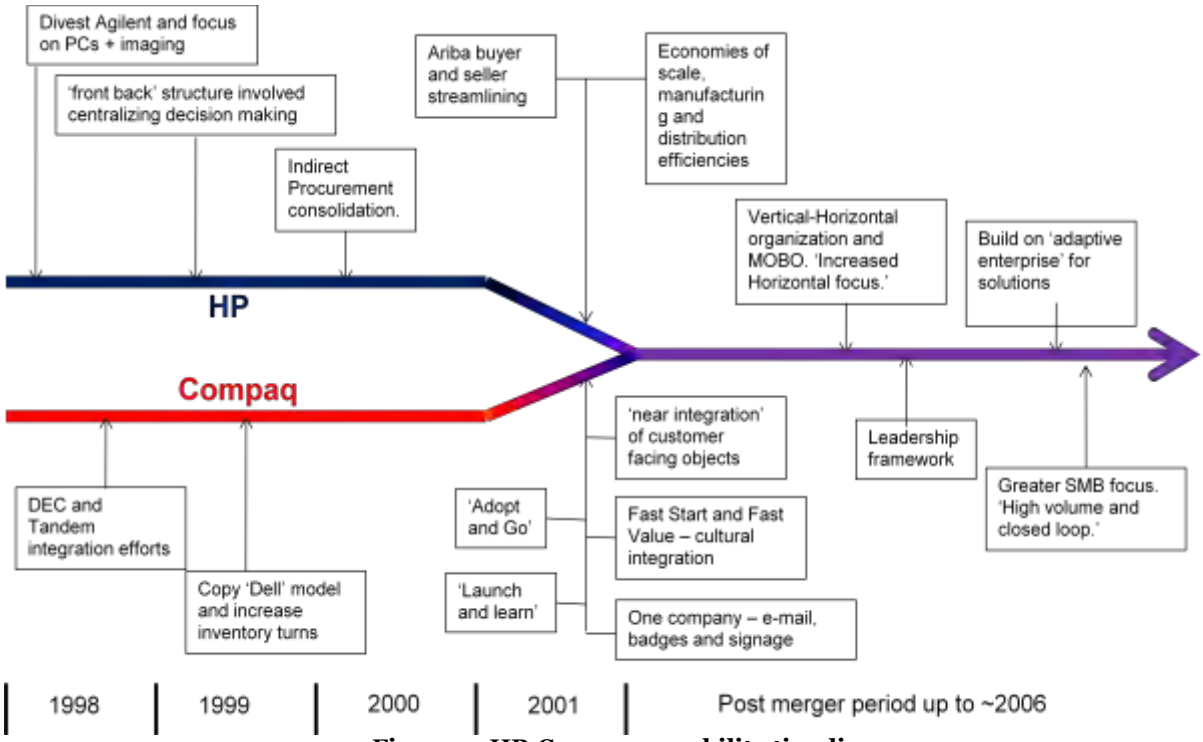

Figure 2: HP-Compaq capability timeline

Adopt-and-Go was a methodology whereby existing HP and Compaq systems devoted to a particular function were evaluated quickly, determining which company had the better system. That system was kept, while others were discarded. Then the clean teams moved on to other decisions. The same process was used for products offered by the merged company.

The process was designed to be streamlined, yielding quick decisions and removing political considerations. One important realization of the clean team was that there was precious little time to study and re-engineer thousands of processes while the rest of the industry was rapidly advancing in products and capabilities. Once the clean room decisions were made, it was up to the rest of the company to implement them. There was no debate allowed, nor reconsideration of decisions. This allowed the merger to proceed quickly and helped yield positive financial performance. It should be noted that the entire process reported directly to the CEO (Carly Fiorina). Any problems in schedule or cost would be noted and reported to her and the executive team on a weekly basis, and they would take appropriate action.

Complementing Adopt-and-Go was the concept of Launch-and-Learn. One of the key mistakes of the Compaq-DEC merger, according to the executives in charge of the HP-Compaq merger, was over-analysis of the possible outcomes of any action. It was decided that there was not enough time to develop high levels of outcome certainty

among all the players involved. Launch-and-Learn embodied the philosophy of taking quick action that was "good enough."

Several other concepts are worth noting. While Adopt-and-Go and Launch-and-Learn relate mainly to technical decision processes and criteria, these other concepts are more socio-cultural in nature, with the goal of managing conflicts. Conflict within the clean team, of course, was a predictable outcome. The concept of "Launch the Moose" was used whereby differences were immediately brought up, discussed and addressed, rather than being allowed to fester and derail the process later. Another concept, "Watch out for Icebergs," was utilized to highlight the potential large-scale problems that could arise, but that had less visibility than the high-profile issues of financials, strategy and product decisions. Finally, "Fast-Start" was a program designed to help employees of the two companies become acquainted and identify and discuss any potential flashpoints.

The overarching goals of the merger related to scheduled launch date and financial performance. Critical business imperatives included the following.

- The merged company operates as "one company from day one" (e.g., a single common email address, identical paychecks, single vision to CompShare, etc.).
- Adopt-and-Go is used for every major business function (e.g., there cannot be a decree that the HP GAAP (Generally Accepted Accounting Principles) reporting for services is the same as for inkjet; the appropriate customer to cash model must be chosen; the anti-trust firewall must be kept in place until merger closing, etc.).
- Maximum cost synergies must be extracted from the merged companies (e.g., premium on time to adoption of a merged supply chain).
- Single company compliance must be in place for all laws and regulations (e.g., SOX (Sarbanes-Oxley), EEU (European Economic Union) labor laws, Export Control, etc.).
- Corporate assets must be protected (e.g., IP management, property control, legal exposure for personnel reductions, contract compliance).

From initial merger discussions to approval of the merger, the clean team engaged primarily in planning, with some prototyping. They could gather information from HP and Compaq employees, but they could not share details of anything that they did. Once the merger was approved, the work started in earnest.

The critical IT systems in the run-up to the merger were the following: Voice Services, Back Office, Data Services, Help Desk, Employee Portal, Client Service, Mail and Messaging Services, Directory Services and Security. Here, the focus is on the email system and the employee portal.

- As of day one, the email system was integrated, mainly via scripts for routing email to the appropriate places. All employees had an email address using the domain hp.com. This also involved merging of the active directory so that all employees were listed. Since authentication was driven from the active directory, this touched a number of other systems.

- The employee portal was unified, mainly via web development, not back-end system merging. This provided all employees with access to personnel records, benefits, stock purchases and HP manuals.

Thus, in the short term, the strategy was mainly to pursue integration for critical systems that must be unified on the "first day." Other systems were selected (or deselected). Some systems were allowed to run in parallel until time came for a business decision to replace the system (e.g., technology upgrade due to obsolescence).

After the merger was consummated, large-scale IT merging and integration started (Basole and DeMillo 2006). A complicating factor here was the sheer number of systems that existed due to prior mergers. There were approximately 70 supply chains in operation at the time of the merger, as well as 35 enterprise resource planning (ERP) systems. Some of these were DEC systems. The goal was to winnow these down to four ERP systems that had code bases to support the three customer-facing divisions (consumer, enterprise and small-medium business), plus support for a number of fulfillment modes and geographic regions.

The goal of migrating from separate legacy HP, Compaq and DEC systems to an SAP system, in particular, was frustrated by organizational silos. The complexity overwhelmed the integration team when the system received an unanticipated surge in orders during data integration. This resulted in a backlog of $120 million worth of orders for enterprise servers. Ultimately, this problem cost the enterprise server division $400 million in revenue and $275 million in profit.

In summary, the following are lessons learned from the HP-Compaq merger.

- There were little or no requirements derived from the business imperatives to drive software implementation.
- The staged implantation of the IT merger – integration of "first day" systems, followed by selection of major systems, was generally successful.
- The Adopt-and-Go selection method coupled with the Launch-and-Learn decision method generated positive results in the highly disciplined, CEO-driven environment. These methods were not formalized for generalization to other situations.
- Unanticipated major changes in the business environment during integration caused disastrous results for part of the enterprise.

- A merger of two enterprises may represent a merger of many underlying systems, some of which were undigested from prior mergers.

4.1.2 DoD Multi-Platform System-of-Systems

In this section, we focus on a Department of Defense (DoD) multi-platform system-of-systems (MPSoS) application where the studied enterprise consisted of DoD Services. The case study to date has been primarily focused on the architectural aspects of the system-of-systems (SoS). In this report, we only overview the issues that occurred as this system-of-systems was developed due to its classified nature.

The architecture of the DoD Multi-Platform System-of-Systems reflected its organization into commands, centers, etc. Incomplete efforts have been previously made to integrate these. The particular SoS of interest for this case study had a basic layered information architecture reflective of the TCP-IP levels. The SoS also relied on and interoperated with many independently-evolving support systems based on different generations of information technology. Hence, there were necessary compromises with respect to the layered architecture to accommodate the legacy systems.

To facilitate SoS development, SysML (Systems Modeling Language) and UML (Unified Modeling Language) (Weilkiens 2008) were the required models for the SoS and its subcontractors. However, a shortage of SE budget and schedule meant that most of the use cases and their architecture representations covered just the sunny-day scenarios. Some midterm redirection and rebudgeting created further constraints. A key scheduling problem was the size of the software to be built. Some cost and schedule models were thus used to prioritize and stretch out the software increments.

To keep up with the new releases, a dedicated group was created. However, the group often fell behind due to the number and complexity of new releases as well as additional constraints. For example, at the SoS level, the different subcontractors would deliver documentation at different times thus causing integration and maintenance problems.

The SoS development and integration processes were further complicated with several external factors:

1. The changing mission capability priorities and evolving multiple stakeholder capability needs meant that there was continuous overlap between decomposing capabilities to requirements and mapping capabilities/requirements to architectures.
2. Subcontractors based their bids on reusing existing assets, which were incompatible with other subcontractors' assets. Interface definition was overly simplistic, focusing on messages vs. protocols.

The SoS developers mentioned several MPTs that would ideally accompany the development and integration processes:

1. There is a need for a support environment that would enable more model-based capture and update of status information (vs. different point-in-time documents), and verification of integration feasibility before beginning component developments.
2. More budget and schedule for baselining the requirements and architecture, and for making necessary revisions once subcontractors are on board.
3. An evolutionary development process with budget and schedule for a continuous systems engineering function that performs analysis and triage of proposed changes, tries to minimize destabilization of the current increment's development, and rebaselines the plans and schedules for the next increment.

The expected outcome of having these improved MPTs, which we plan to propose and design in our future work, include avoidance of scalability problems, improved change management, better understanding of budget and effort relative to the proposed SoS size and costs, improved value prioritization for both full operational capabilities and initial increments.

4.1.3 Regional Area Crisis Management System-of-Systems

The Regional Area Crisis Response System-of-Systems (RACRS) is an example of integrating multiple existing systems (Lane and Bohn 2010). This case study focused on a notional SoS that is based in part on actual systems and capabilities deployed in Southern California. In essence, RACRS is a temporary integration of a number of disparate agencies (e.g., Police, Fire department, Satellite Imaging Systems) in response to various regional crisis events (e.g., fires, hazardous material spills, terrorist activities).

The development of RACRS has been motivated by the communication challenges between the different local agencies with their own proprietary IT systems that often resulted in very weak or no integration. Consequently, the goal of RACRS has been to allow the local agencies to independently operate outside of the SoS and then quickly dynamically reconfigure and join the regional SoS in response to an incident.

Overall, the RACRS case study served as an example of developing a system-of-systems that consists of a number of independent agencies, where each agency should be able to autonomously join and leave the SoS. The technique presented in (Lane and Bohn 2010)) suggests that RACRS and other systems similar to RACRS can benefit from explicit decomposition of high-level capabilities to agent-oriented requirements with explicitly managed traceability.

4.1.4 Health Care IT System

The Health Care IT System is an example illustrating typical interoperability issues. To provide some background, consider medical providers, hospitals and hospital systems who have adopted information technology to handle patient records and support clinical

decision-making. In this setting, a variety of vendors have entered the market, and different COTS products have proliferated. These products are generally not interoperable. Thus, when a patient sees multiple providers for different types of treatments, each provider can access only a limited part of the patient history – the part housed in that provider's silo. Consequently, significant efforts have been made in recent years to develop interoperability standards for existing and new COTS products.

The system on which we focused in this case study was initially a stovepipe (i.e., standalone) hospital health care system with integrated subsystems such as Electronic patient records, Pharmacy, and Laboratory. After more than ten years of operation, the single-hospital system "opened up" to interoperate with multiple hospital sites and other hospital systems. The legacy standalone system continued to operate (at least to some extent) as various capabilities in it were retired and transferred to other more modern systems, COTS products. Figure 3 shows a conceptual architecture for the system. Further details are available in (Suri 2009).

Figure 3: Conceptual architecture for the health care IT system

The transformation from a legacy system to a modern SOA-based architecture was incremental, where new capabilities were provided to the developer as they were identified. To determine the next necessary increment, the approach (i.e., what changes to what subsystems are required) and cost/schedule estimates were provided to the customer with the capability being added to the product "backlog." As planning would begin for the next release, the customer would re-evaluate the priorities associated with everything in the product backlog and decides what to fund in the next increment. In addition, once the changes are planned to be made to subsystems that have some outstanding low priority changes/capabilities in the backlog, those were often included with the higher priority capability for cost reduction (it is costly and increases risks if you "open up" software modules for a few minor changes, and is more cost effective to combine these lower priority changes with high priority ones that are going to require complete regression testing of the module/subsystem).

The non-technical setting of the health care system included a fixed staff in place for maintaining each health care subsystem. Furthermore, the stringent safety requirements (e.g., patient safety) imposed limited programmer mobility where some general programmers were allowed to move from subsystem to subsystem, while the key subject matter experts and software experts for a given subsystem did not move around.

The health care study also provided some unique insights in terms of the effects of the funding structure as the program (a US government program) was large enough to be a Congressional line item and its budget depended on Congressional appropriations. Therefore, the portion of the product backlog that was worked off in a given increment depended on the volatile external budget with very limited control.

To integrate the health care system into a broader SoS, the main health care system was opened up to share information with other health care systems and health care sites, the HL7 interface protocol, a standard with respect to health care systems, was selected (as well as supported by many COTS health care systems). However, there were several examples where this was not sufficient. A classic type of problem occurred early on in integration when one system treated midnight as "0:00:00" and another system treated midnight as "24:00:00," causing shared patient record entries to be incorrectly sorted.

4.1.5 BACK-OFFICE IT SYSTEM

The Back-Office IT System (BOITS) refers to an example integration of office IT systems performed for an Internet Service Provider (ISP) start-up. The primary challenge in the development of BOITS was that the basic capabilities had to be implemented within approximately 6 months in order for the new business to start selling services. The initial business area capabilities were developed through brainstorming with subject matter experts and personnel being hired to manage the various business areas for this start-up company. The identified primary capabilities included order entry, product/service management, customer care, billing, network monitoring, and several others. The time constraints pushed the development into a purely COTS-based solution.

The selection of appropriate COTS for BOITS included separate assessment of "best of breed" solutions for the different capabilities. Initial capabilities were then compared to various COTS products in the market as to how well they were accommodated: available in the out-of-the-box product, available with some custom development work, not available at all/requires new development. For those that required some/all custom development to be integrated with the COTS product, at least two alternatives were considered: 1) have the vendor develop the desired functionality and integrate into the next version of the COTS product; or 2) have the integration organization develop the desired new functionality and determine the total cost of ownership of this software, given that it may need to be updated/re-integrated with each new COTS upgrade. These two options were used to prioritize capabilities and conduct trade studies across the known COTS products.

However, the strategy of selecting the "best quality" COTS a priori also raised significant issues as the "best quality" COTS suffered from many compatibility issues. For example, out of about 40 COTS products, there were around 30 different formats for "customer

address" that had to be reconciled/converted. In the end, the selection of COTS for BOITS had to be based on the quality of the candidate COTS, how they can be integrated, and the analysis of the existing COTS trade studies. Note that as a result, some of the initial "high priority" capabilities and COTS were deemed to be too expensive from the perspective of the total cost of ownership.

Once the COTS products were selected, efforts began on developing a detailed integration strategy: point-to-point for some products, service-oriented approach for others, and an underlying data repository/warehouse for maintaining all of the COTS data. One of the key issues to deal with in this integration was standardizing data formats across all of the COTS products. The system integrator was tasked to provide cross cutting capabilities such as integrated user interface/single point of data entry, sales force automation, and security/access control.

4.1.6 FBI Virtual File Case System

In the 1990s, it became obvious that the Federal Bureau of Investigation (FBI) had an antiquated system of tracking case file information and evidence. Most of the problem centered around the difficulty of sharing information between different investigations that were pursuing related cases. As a result, in 2000, the FBI inaugurated the design and implementation of a new case information management system, what eventually became known as the Virtual Case File (VCF) system. The VCF became one of the more famous software failures in history. Further documentation of the case study is available in Goldstein (2005).

Historically, the Bureau has consisted of a number of different divisions, ranging from criminal investigation, to counter-intelligence, to law enforcement services (e.g., labs and testing). Up until 2004, each division had its own IT budget and systems. This led to approximately 40 to 50 different investigative databases across the divisions, with significant duplication of function and information. The FBI did have an automated system for storing case information, the Automated Case Support (ACS) system. This system was obsolete, though, and suffered numerous shortcomings. Fundamentally, it did not effectively support the linking and sharing of information. It was Bureau policy that all official records were to be entered into the ACS. However, this included only official FBI forms and, for instance, not handwritten notes. Many agents were not aware of its capabilities beyond the indexing of documents.

Thus, the goal of the new system was to replace the ACS, integrate the existing disparate database systems, and convert any existing paper case files to a fully automated system that would better support information sharing across the Bureau. The VCF started as the User Applications Component portion of the Trilogy project, a major effort to upgrade FBI information technology that also included computer hardware and network upgrades. After 9/11, it became the VCF, with a new set of prescribed capabilities to

focus specifically on information sharing. The intent was to address the failures in information sharing that played a role in not detecting terrorist plots.

Unfortunately, the project was plagued with problems from the start. One problem was cultural, in that the FBI had never had an institutional focus on software systems or software development. In fact, many agents were reluctant to use existing IT systems. This played into the specific dysfunctions and failures of the VCF. A detailed treatment of the design and development process for the VCF is beyond the scope of this report. Nevertheless, the outcome is of interest.

On December 13, 2003, the contractor delivered the VCF, and it was rejected by the FBI (by then with new IT management), which cited numerous deficiencies. The FBI did, however, approve further development and testing of the workflow component of the VCF, which functioned relatively well. Rechristened the Initial Operating Capability (IOC), the workflow component was field-tested and found not to improve user productivity, in large part because it still interfaced with the legacy ACS. For example, the IOC automated many processes associated with filing reports and monitoring their approval. The ACS still required printed reports and handwritten approval signature, causing redundant user work. The IOC was not deployed further.

The following are lessons learned from the VCF in terms of net-centric requirements management.

- Poor planning. The FBI lacked an enterprise architecture blueprint. Thus, there was little documentation on existing missions and processes and how technology is used and structured to support them, much less a roadmap to help guide future decisions about IT.
- A major change in scope after 9/11. What was originally posed as a replacement for ACS became the VCF, with a focus on information sharing to address shortcomings that contributed to the terrorist attacks.
- An overly-detailed requirements document. The requirements document was written at a very low level and did not map to user needs.
- Significant changes to requirements. The review process facilitated the submission of requirements changes by users, and there was little or no analysis as to the cost or schedule implications of these changes.
- Ambitious schedule. A major schedule acceleration without accompanying analysis regarding functionality and cost.
- Reluctance to change architecture in response to major change in scope.
- Asynchronous development schedules between the hardware and network components of Trilogy, on the one hand, and the VCF on the other. Delays in hardware and network development meant delays in testing the VCF on a real system.
- Lack of software communications standards to guide development.

- Lack of a transition plan.
- Lack of process/tools to transition paper records to automated system.
- Unanticipated issues involving integration with legacy systems. The IOC was field-tested in an environment where it was effectively integrated with the ACS. The duplicative nature of system functions required redundant user actions, which impaired productivity.

4.2 INTEROPERATION, INTEGRATION AND MERGING

Systems can be joined in a number of ways. Three methods considered here are interoperation, integration and merging. Each is appropriate for different situations, typically involving duration, system purposes, and technical difficulty associated with conflicts between systems.

Figure 4 illustrates these concepts, where an integrated system refers to both interoperation and integration.

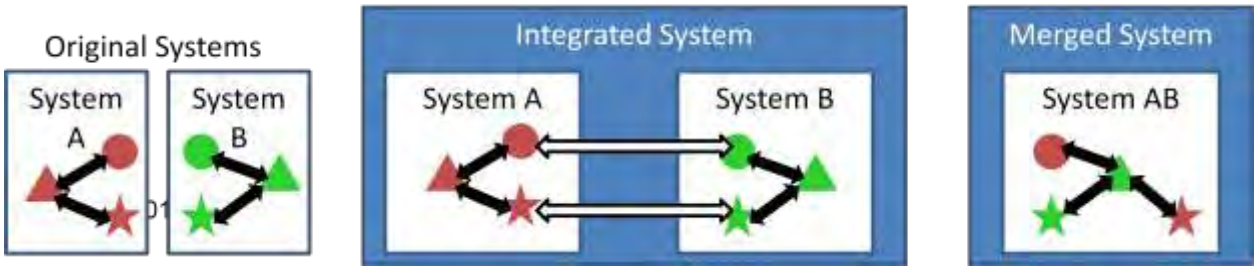

Figure 4: Types of system joins

Under interoperation, constituent systems are assumed to have a high level of interoperability, which facilitates their joining together with minimal integration effort. Thus, interoperation works well for temporary join situations, since the systems can be decoupled in a straightforward manner. However, interoperability is a significant up-front investment for a constituent system. In net-centric enterprises, the specific joins typically are not predictable far enough in advance to justify this investment solely for the purpose of a system join.

Under integration, interfaces must be developed. This can involve substantial effort and may not be justified for short duration joins. In a merger, the strategy is to reuse the best parts of the constituent systems and combine them into a new system. Such an approach is best suited to a permanent joining of systems due to the difficulty of separating the systems once merged. Table 2 summarizes these three types of joins. While they are shown as three distinct alternatives, it should be understood that there is a continuum of possibilities between a "full" interoperation and a "full" merger.

Interoperation	Integration	Merging
Original systems retain their own identities as subsystems of the integrated system.	Original systems retain their own identities as subsystems of the integrated system.	Original systems' identities are lost; however, some of the original systems' subcomponents may exist in the merged system.
All functionality in the original systems is retained (although some of it may not be used).	All functionality in the original systems is retained (although some of it may not be used).	Some functionality in the original systems may be lost or discarded.
Involves using inherent interoperability of constituent systems.	Involves creating, using, or adapting externally exposed interfaces of the original systems.	Involves changes to system internals.
Temporary	Temporary-to-permanent	Permanent

Table 2: Types of system connections

Land and Crnković (2011) add two other alternatives in their analysis of ten case study system mergers/integrations that occurred mainly in industry – choose-one and start-from-scratch. In their analysis, they also use loose integration, which encompasses both integration and interoperation. They find that organizations sometimes adopt blended strategies in joining systems. Additional findings include:

- Integration is enabled when the constituent systems are oriented around databases, and when data transfer and synchronization capabilities are sufficient.
- In theory, reusing the best parts of the constituent systems and forming a merged system should reduce cost and implementation time while helping ensure proven quality. This is not necessarily the case in practice, though.
- There must be sufficient similarities in architectures (structure, data model and technology) to make merging feasible. In addition, the case for merging is aided when each constituent system has unique capabilities.
- Choose-one is enabled when one system has better technology than the other(s), especially when this system addresses most of the functionality of the other system(s).
- Start-from-scratch should be investigated if all constituent systems are nearing obsolescence, or if new capabilities are desired that no system addresses.

Table 3 highlights key aspects of the case studies considered in this report relative to the types of system joins employed.

Case Study	Key Aspects
Health Care IT	System integration • There was significant use of different COTS products among different hospital sites/systems. • Each hospital site or hospital system had its own IT system (i.e., sub-system relative to overall IT system). The overall IT system was to interoperate with these heterogeneous sub-systems. • A fixed staff was in place for maintaining each health care subsystem. • The healthcare industry has emerging interface protocols for interoperability. One of these was used (HN7), but did not prevent all problems. • In one instance, midnight was treated as 0:00:00 by one sub-system and 24:00:00 by another, resulting in sorting problems for patient records. • Duration was likely to be permanent.
DoD Multi-Platform SoS	System integration • The enterprise was a DoD service organized into many different and heterogeneous commands, centers, etc. • There were many independently evolving sub-systems that relied on different technologies.
HP-Compaq	First-order integration, then merger via selection • One imperative was that the two be "one company from day one." Integration was used to achieve short term unification. • At the business process level, the merger consisted of choose-one among multiple possible system choices using the adopt-and-go method.
Back-Office IT System	System integration using COTS • There was significant schedule pressure. • Reconciling data formats was a major issue.
FBI Virtual Case File System	Start-from-scratch and system integration • The initial VCF concept was a start-from-scratch system that would replace the obsolete ACS and other disparate IT systems • When VCF was terminated, part of it was rechristened as IOC and was integrated with ACS.

Table 3: Case studies -- merging vs. integration

4.3 CONTEXT

The context in which integration and merging takes place is important to understand, since it impacts the potential costs, schedule and functionality of the unified system. Here, several context variables are identified, and for each variable, several possible values exist. These variables are descriptive in nature.

- Type. The type of integration or merging describes the level at which the integration or merging takes place. Possible types include the following, in order of increasing complexity and difficulty.
  - User interface
  - Data
  - Control
  - Process

- Orientation. Systems exist at different levels, and within levels they may exist in different organizations. Systems may be integrated or merged that exist at different levels, or that exist in different organizations at the same level. Of course, both are possibilities. Thus, possible values for orientation include the following.
  - Horizontal
  - Vertical
  - Both

- Duration. The integration or merger has a duration, which may or may not be known up front. Thus, possible values for duration include the following.
  - Temporary/transient and of known duration
  - Temporary/transient and of unknown duration
  - Permanent and known to be permanent

- Multiplicity. The integration or merger may feature two or more systems. In some instances (e.g., HP-Compaq), what seems to involve two systems on the surface actually involves many others, due to past integrations and mergers.

- Concurrency. An integration or merger may operate as a single activity with a definitive beginning and ending. Sometimes, there may not be a definitive ending. Other times, in mid-unification, a new system may be introduced for integration or merging.

Table 4 illustrates how key case studies are categorized by integration/merging type. One key finding from this analysis is that data integration/merging involves significantly more challenges and risk than user interface integration/merging.

Case Study	Key Aspects – Type
Back Office IT System	User interface • System integrator was tasked to provide a "single point of data entry" for the system.
Health Care IT System	Data • The primary purpose was to share patient records across multiple systems.
HP-Compaq	Process-Control • The Adopt-and-Go selection method effectively merged the process and control aspects of systems. Data • Data integration occurred for ERP systems integration.
FBI Virtual Case File System	Data • The primary goal was to integrate case file data from a number of different sources (40-50 databases).

Table 4: Case studies -- integration/merging type

Case studies illustrating the notion of orientation are shown in Table 5. From these examples, horizontal orientations tend to be suited toward realizing capabilities, while vertical orientations tend to be suited toward interoperating.

Case Study	Key Aspects – Orientation
HP-Compaq	Horizontal • Adopt-and-Go was used for every major business function.
FBI Virtual Case File System	Horizontal • The primary effort was to convert paper-based and disparate existing case file systems to a single automated system.
Regional Area Crisis Management System	Vertical • The goal was to provide interoperability among independent systems at different levels during a regional crisis.

Table 5: Case studies -- integration/merging orientation

Table 6 shows case studies that illustrate the key concepts involved in duration of integrations and mergers. One note of interest is that unwinding of an integration or merger most likely requires significant up-front planning to be successful. For instance, each system in the Regional Area Crisis Management System would need interoperability as a feature that can be enabled and disabled. While the HP-Compaq system merger did not proceed beyond the planning stage prior to merger approval, there would have been significant up-front planning needed if there were a chance that the unwinding would have occurred after implementation had started. However, the

possibility of the merger being derailed did, in fact, compress the time available for implementation. This led to the imperative that "clean room" decisions were to be executed and could not be changed.

Case Study	Key Aspects – Duration
Regional Area Crisis Management System	Transient/temporary of unknown duration • RACMS consisted of independent systems designed to interoperate in response to various regional crisis events. These events were typically of unknown duration.
HP-Compaq	Permanent • The intent of the merger was to be permanent. However, at any point the merger could have been killed and the merger process would have to be unwound. This would have primarily involved unwinding plans.

Table 6: Case studies -- integration/merging duration

Finally, Table 7 shows key aspects relating to concurrency. Custom, pair-wise solutions are most likely sub-optimal when dealing with significant concurrency.

Case Study	Key Aspects – Concurrency
Multi-Platform System-of-Systems	Multi-system, unordered concurrency • The system architecture reflected organizational units of commands, centers, etc. • Each system evolved separately, requiring continual demand on overall interoperability.
Regional Area Crisis Management System	Multi-system, unordered concurrency with systems leaving and entering • Each system entered the system-of-systems in response to a particular crisis and could leave it once the crisis was over (with the possibility of rejoining later). • Each system evolved separately, requiring continual demands on overall interoperability.
HP-Compaq	Multi-system, ordered concurrency • "First day" systems were integrated first. • Major systems were selected from up-front. Then the implementation of selection occurred over time. • Some systems operated side-by-side until replacement as a business decision (i.e., better technology).

Table 7: Case studies -- integration/merging concurrency

4.4 CONSTRAINTS

Of course, systems integrations and mergers are usually faced with any number of constraints. Here, the possible constraints are classified. Constraints can serve to help prioritize decisions with respect to risk, cost or schedule. The constraints studied here include the following.

- Platforms and technology
  - Hardware, OS, programming language, etc.
- Architectural style
  - Client/server, pub/sub, etc.
- Information access
  - Availability of source code, architecture documentation, etc.
- Cost and schedule
- External constraints

Table 8 shows critical issues from the case studies involving technology constraints. There may not be an optimal integration/merging strategy if there are significant technology constraints.

Case Study	Key Aspects – Technology Constraints
Multi-Platform System-of-Systems	• SysML and UML were the required models. • Compromises were made to accommodate the legacy systems.
Health Care IT System	• There was integration with many existing/evolving systems.
Back Office IT System	• There were many compatibility issues with COTS systems.
FBI Virtual Case File System	• There were 40-50 existing case file databases.

Table 8: Case studies -- technology constraints

Architecture constraints may exist due to legacy systems or to incompatibilities between architectures of systems to be integrated. Issues involving architecture constraints are highlighted in Table 9. Architectures not suited for a newly integrated system, or conflicts between architectures of systems to be merged/integrated tend to escalate cost and may reduce functionality.

Case Study	Key Aspects – Architecture Constraints
Multi-Platform System-of-Systems	• The previous layered architecture was reused. • Compromises were made to accommodate the legacy systems.
HP-Compaq	• Merging of the active directory touched a number of other systems, since virtually all authentication was driven off the active directory.
FBI Virtual Case File System	• There was a reluctance to change the system architecture after a major change in scope following 9/11.

Table 9: Case studies -- architecture constraints

Information access constraints exist in many integration/merge situations due to lack of documentation or available subject matter expertise. Table 10 shows issues involving information access constraints in the case studies. Information access is impaired due to both technical and organizational causes.

Case Study	Key Aspects – Information Access Constraints
Multi-Platform System-of-Systems	• Information was limited due to the nature of the multiple/evolving systems.
Health Care IT System	• The various systems being integrated were located in different hospitals and different hospital systems. Key subject matter experts and software experts for a given system generally do not move around.

Table 10: Case studies -- information access constraints

Cost and schedule constraints are ubiquitous in integration/merger situations. The effects of cost and schedule constraints on selected case studies are described in Table 11. Clearly, cost and schedule drive prioritization of capabilities, decisions and activities. The extent to which this is done effectively in large measure determines the success of the system integration/merger.

Finally, system integrations and mergers are often at the mercy of external constraints, which may change during the integration/merge process. Table 12 illustrates some key types of external constraints imposed on the case study systems.

Case Study	Key Aspects – Cost and Schedule Constraints
Multi-Platform System-of-Systems	• A shortage of systems engineering budget and schedule meant that most of the use cases and their architecture representations covered just the sunny-day scenarios.
Health Care IT System	• Invariably, there was a backlog of feature requests for the integrated system. How much of the backlog was worked off in a given increment depended on the available budget.
HP-Compaq	• The "one company from day one" imperative imposed a schedule constraint on the first-order systems integration. • Cost constraints were imposed due to the imperative that the business merger objective was substantial cost consolidation. • Any variances on cost and schedule were reported immediately to the CEO level.
Back Office IT System	• Due to time constraints, COTS solutions were pursued.
FBI Virtual Case File System	• A major schedule acceleration was approved without analyzing the effect on capabilities/features and cost. These three factors are interdependent, and one cannot be changed without affecting the others.

Table 11: Case studies -- cost and schedule constraints

Case Study	Key Aspects – External Constraints
Multi-Platform System-of-Systems	• The subcontractors would deliver documentation at different, unpredictable times.
Health Care IT System	• The budget was unpredictable, as it depended on Congressional appropriations.
HP-Compaq	• The merging companies had to comply with anti-trust regulations, which meant that no actual system integration or merging could occur until the business merger was approved.
FBI Virtual Case Files System	• In response to 9/11, it was decided to have a major change in the scope of the system. • There was not an existing hardware set on which the system could be tested, since VCF was part of the Trilogy project that also featured concurrent hardware development. Delays in hardware deployment meant delays in VCF testing.

Table 12: Case studies -- external constraints

4.5 DECISION FRAMEWORK

Decision-making in the context of the net-centric ecosystem – especially as it relates to the capabilities, requirements, and architecture of a system created through interoperation, integration, or merging – requires simultaneously considering multiple different factors and balancing the interests of multiple stakeholders.

The most fundamental trade-off that must be considered is the cost of undertaking a decision vs. the value returned. In most cases, it is not practical or technically feasible to quantify the cost and value precisely. For example, in addition to the financial cost a decision may incur, it may also have costs in the form of opportunity costs, risk of failure, and delayed deployment. The value of a decision may depend on subjective and unpredictable considerations such as the value provided to the warfighter, the occurrence of a low-probability situation (such as a catastrophe or disaster), or the emergence of an unexpected threat.

Although cost and value are difficult to determine exactly in the context of net-centric interoperation, integration, and merging decisions, they can be characterized qualitatively and ranked relative to each other. Even an informal (but carefully considered) characterization of cost and value can be useful to decision-makers. As shown in Figure 5, if the cost and value of a set of activities or decisions can be ordered relative to each other, the relative priority of each activity or decision is given by its location in the two-dimensional space shown. Decisions falling in the upper-left quadrant are the highest priority, while decisions falling in the lower-right quadrant are the lowest priority. We refer to this two-dimensional space as the *prioritization plane*.

While concept of the prioritization plane is somewhat informal, it forms an important basis for our proposed methodology and future research. The use of the prioritization plane is discussed further in Section 5.2 Integrated Methodology.

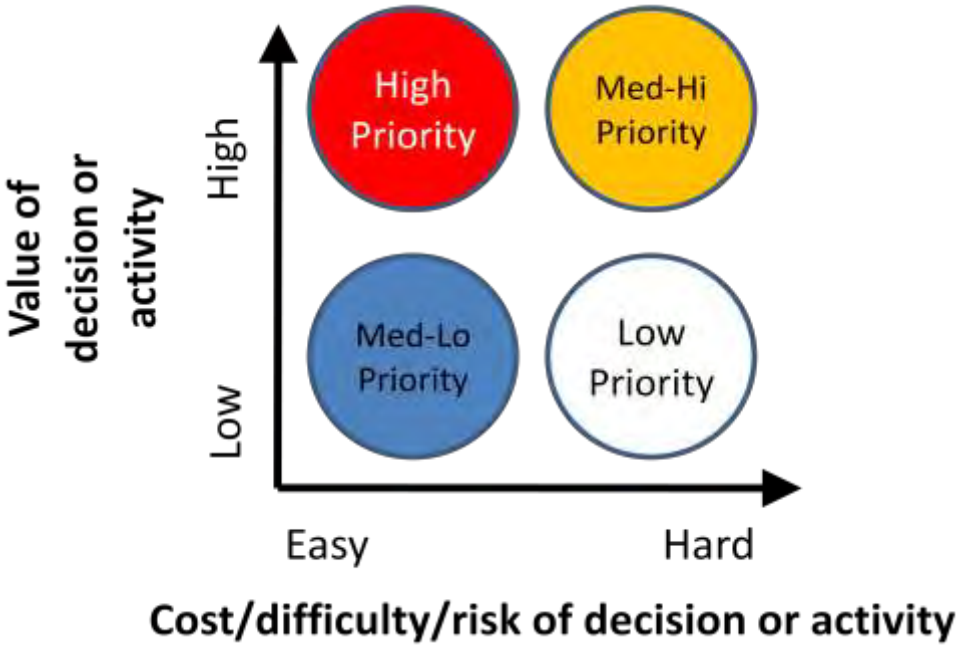

Figure 5: Two-dimensional plane depicting the value of decisions by value and cost

5 METHODOLOGY SPECIFICATION

Based on the case study analysis, as well as generic study of the problem, this section proposes a methodology for addressing the problem of requirements management in a net-centric enterprise. This problem encompasses a number of sub-problems. Due to a number of factors, including the net-centricity of the enterprise, its evolution over time, and the resulting impacts on capabilities, functions and requirements, the methodology is iterative in nature. In addition, we propose the use of component MPTs to address the sub-problems. Component MPTs already exist for some of the sub-problems, but typically will require enhancement to address the full spectrum of needs in the net-centric enterprise.

5.1 COMPONENT MPTS

The individual component MPTs are presented first. The next section addresses how they fit into an overall methodology.

5.1.1 WINWIN NEGOTIATION MODEL

The primary goal of this MPT is to help the discovery, negotiation, and reconciliation of capabilities and requirements in a highly collaborative, interactive, and interdisciplinary negotiation process that involves heterogeneous stakeholders. The WinWin approach (Boehm, Grunbacher et al. 2001) involves having a system's success-critical stakeholders participate in a negotiation process so they can converge on a mutually satisfactory or win–win set of requirements.

WinWin has been defined as "a set of principles, practices, and tools, which enable a set of interdependent stakeholders to work out a mutually satisfactory (win–win) set of shared commitments" (Boehm, Grunbacher et al. 2001). In this definition, *interdependent stakeholders* can be people or organizations. *Mutually satisfactory* generally means that people do not get everything they want but can be reasonably assured of getting whatever it was to which they agreed. *Shared commitments* are not just good intentions but carefully defined conditions. If someone has a conditional commitment, he or she must make it explicit to ensure all stakeholders understand the condition as part of the agreement.

The win-win negotiation model consists of four types of artifacts – Win Condition, Issue, Option, and Agreement (Figure 6), which are used supported by the tools developed at USC over the period of the last ten years.

- Win Condition – captures individual stakeholders' desired objectives.
- Issue – captures conflicts between win conditions and their associated risks and uncertainties.
- Option – identifies candidate solutions to resolve an issue.

- Agreement – captures shared commitment of stakeholders with regard to accepted win conditions or adopted options.

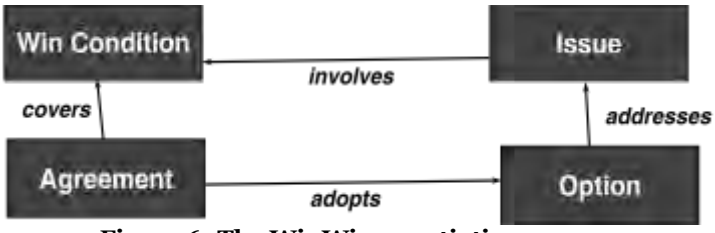

Figure 6: The WinWin negotiation process

The particular approach that we have evolved includes a WinWin negotiation model for converging to a win–win agreement and a Win-Win equilibrium condition to test whether the negotiation process has converged. The negotiation model guides success-critical stakeholders in elaborating mutually satisfactory agreements. Stakeholders express their goals as win conditions. If everyone concurs, the win conditions become agreements. When stakeholders do not concur, they identify their conflicted win conditions and register their conflicts as issues. In this case, stakeholders invent options for mutual gain and explore the option trade-offs. Options are iterated and turned into agreements when all stakeholders concur. The stakeholders are in a WinWin equilibrium condition when the agreements cover all of the win conditions and there are no outstanding issues.

The latest tool we developed to support WinWin negotiation is a wiki-based negotiation support tool WikiWinWin (Wu, Yang et al. 2009). Besides the win-win equilibrium theory, the underlying software development theory is the Value-Based System and Software Engineering (VBSSE) framework (Boehm and Jain 2005). The WikiWinWin creates a sequence of steps and instructions to guide the stakeholders working out mutually satisfactory requirements. During each step, the system displays one or more tools with which the team can generate, organize, and evaluate concepts and information.

The WikiWinWin negotiation process begins with setting up the negotiation context, proceeding to negotiate the win conditions, issues, options, and agreements, and continuously refining the negotiation as a project proceeds. The negotiation results will be used to generate the System and Software Requirements Description.

Setting the WikiWinWin negotiation context involves the tasks of (1) identifying, engaging, and instructing stakeholders, (2) holding a stakeholder kick-off meeting, (3) define terminologies and requirements related concepts, and (4) reviewing and expanding negotiation topics. WikiWinWin then guides the stakeholders in identifying goals and preferences, identifying and resolving conflicts, prioritizing requirements, and achieving mutually satisfactory agreements. Specifically, the stakeholders can use WikiWinWin in support of (1) brainstorming win conditions, (2) converging and

surveying on win conditions, (3) agreeing on win conditions or identifying issues, (4) providing options, and (5) reaching agreement. To support continuous refinement and evolution, WikiWinWin provides facilities for browsing the recent changes.

Currently, WikiWinWin is designed for the requirements reconciliation of a single system. Our future work, including the efforts planned for the continuation of this project, will incorporate SoS engineering facilities into WikiWinWin

5.1.2 Enterprise Transformation Framework

Enterprise transformation refers to the action by an enterprise to change its mission, capabilities, functions and operations to address perceived value deficiencies. A value deficiency has multiple meanings. For instance, in the business world, it could refer to a significant reduction in market share and revenue that should be addressed via cost reduction and/or a change in product strategy. It could also refer to new opportunities that require new, currently non-existent capabilities. In government, it could mean changes in the external threat environment that constitute changes in mission, requiring transformation of processes, functions and requirements to meet the new mission.

Enterprise transformation can be facilitated via IT integration, via transformed work and information processes, or via changes in strategy. Enterprise transformation can be characterized within a framework that considers scope, ends and means (Rouse 2006). This framework is depicted in Figure 7.

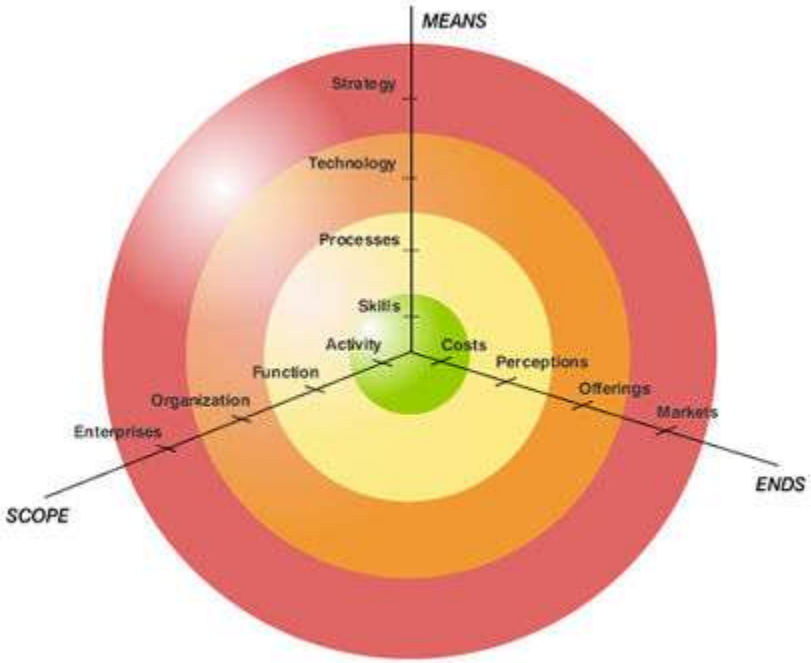

Figure 7: Enterprise transformation framework

This framework can be considered as a risk and opportunity tool. Staying in the green and perhaps the yellow parts of the transformation framework tends to enable continued competitiveness through better performance, with relatively little risk.

Attempting change in the orange and especially the red parts of the framework involves attempting to change the game. The potential impact is enormous, but the probability of success can be quite low. True innovating organizations pursue change at the outer edges of the framework, but often fail. Organizations successful at transformation are forced to pursue the first choice, often by the innovators. Successful innovators change the world and have enormous impact, while successful transformers stay in business and make reasonable impact.

5.1.3 ADOPT-AND-GO SELECTION

In a business merger situation, the organizations involved typically have systems in place for all required capabilities and processes. The question, then, is whether to merge them, integrate them, or start-from-scratch. In the HP-Compaq merger, which occurred under significant time duress, this was a major issue. They adopted a very pragmatic approach to solving the problem. This approach is abstracted and formalized here as Adopt-and-Go.

Essentially, the enterprise and its constituent organizations have a number of business processes. The processes enable capabilities. The processes are nested in the sense that high-level processes have "black-boxes" that essentially invoke lower level processes for results. Thus, there is a hierarchy of processes.

At the same time, the enterprise and its constituent organizations have numerous IT systems dedicated to fulfilling the various missions and business processes of the overall enterprise, as well as of the organizations themselves. The fundamental process is one of selection of IT systems for specific processes, as shown in Figure 8.

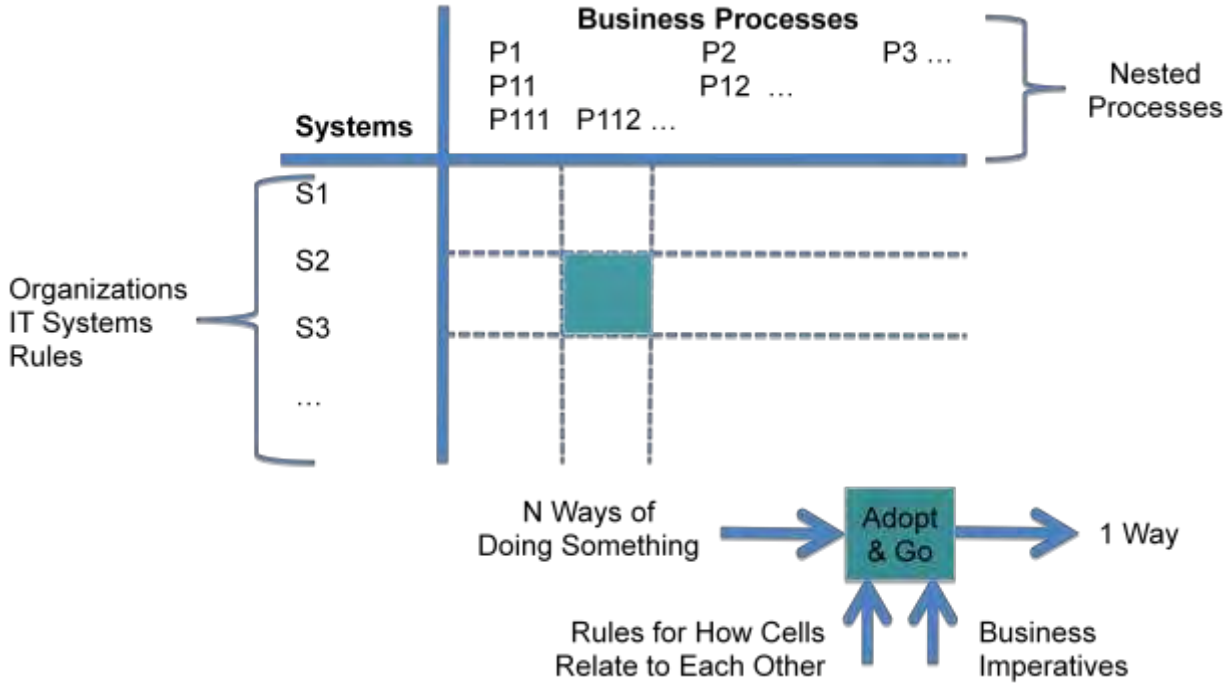

Figure 8: Adopt-and-Go

Clearly, any selections must conform to the business imperatives driving the merger or being continued upon merger completion. Selection decisions may also impose constraints on other decisions via the rules by which various IT systems and business processes relate to one another. These constraints must be understood as the decision process unfolds. In the HP context, Adopt-and-Go decisions were made separately from the operating business, which was then tasked with implementing them without the option of requesting a re-evaluation or change.

5.1.4 SysML-Based Capability Engineering

Our previous work has proposed a process of mapping system-of-systems (SoS) capabilities to functions of constituent systems (CS), while using SysML as a modeling paradigm (Lane and Bohn 2010). We summarize the mapping and the SoS modeling processes next. The overall process is shown in Figure 9.

The system or SoS modeling process starts by setting the context: understanding what is in the SoS, and what is not. The context also includes who and what will interact with the SoS, and what information will be passed to and from the SoS. The context for each CS can be modeled in a SysML context diagram.

The next step in the modeling process is deriving the top level services (i.e., the required system capabilities) that the SoS provides to the environment and other external entities (actors). This list of services is analyzed and refined by the stakeholders and analysts from the engineering team. The engineering team then works on the Use Case

Specifications that describe the major actions necessary to perform the use case and all of the alternate actions.

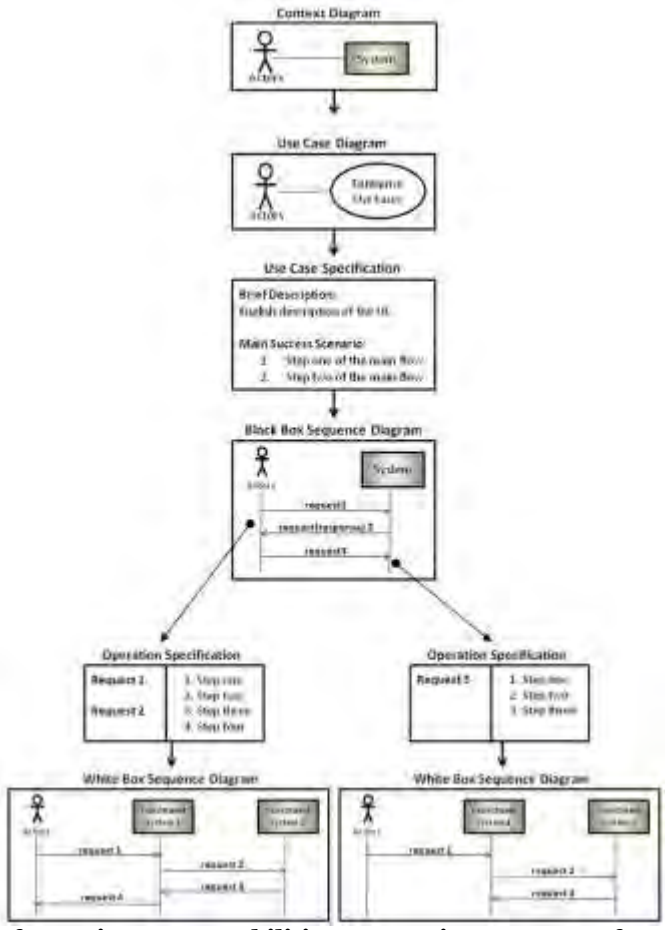

Figure 9: The process of mapping SoS capabilities to constituent system functions (Lane and Bohn 2010)

Each action that requires an interaction between the environment and the SoS forms the basis for a request on the Black Box Sequence Diagram. Next, the Black Box Sequence Diagrams are built to show the flow of requests that pass between the SoS and the environment. These requests form the basis for the SoS-level operations.

Subsequently, each SoS-level operation is transformed into an Operation Specification. The Operation Specification documents the actions necessary to complete the operation and describes the interactions between the constituent systems and the external entities. The Operation Specification can be modeled as a SysML object block.

These actions specified in an Operation Specification form the basis for the White Box Sequence diagrams. These are intended to depict the flow of requests between the CSs. This process of decomposition using the Operation Specifications and White Box

Sequence diagrams continues until the level of detail required to generate a solution is reached. Note that while these steps are described in order, parts of this process are done in parallel and often in iterations.

To summarize, when first starting to model an SoS, it is important to start at the highest level, first viewing the SoS as a black box and focusing on the SoS capabilities and associated external inputs and outputs, then working down to lower levels of detail by replacing the SoS-level black box with an SoS white box that represents each of the CSs as a black box. Then those CS black boxes can be translated into CS white boxes, as needed to better understand CS capabilities and functions.

5.1.5 DoD Systems Engineering Guide for Systems-of-Systems

The DoD Systems Engineering Guide for Systems-of-Systems (DoD 2008) codifies the primary challenges in SoS engineering and identifies best practices distilled from studies of eighteen SoS engineering efforts. The information captured in the DoD SE Guide for SoS is related to the research described in this report because SoS engineering is heavily dependent on effective management of capabilities and requirements and integration of complex systems. In particular, Section 4.1.1 – Translating Capability Objectives and Section 4.1.6 – Addressing Requirements and Solution Options describe MPTs relevant to the research challenges addressed by this report.

As described in Section 4.1.1 of the Guide, the process of translating capability objectives (1) articulates and codifies the high-level expectations for a SoS and (2) refines a set of requirements for meeting those expectations. To accomplish these goals, the Guide identifies the following best practices:

- Defining "variability in the user environment which will impact the different ways...functions will be executed";
- Using reference missions and use cases to "evaluate the operational utility of the SoS";
- "Working with the SoS manager, users, and stakeholders" to develop "understanding of priorities and relationships";
- Tracking "the dynamics of change as they influence the SoS objectives and expectations";
- "Separating objectives from systems" by avoiding all "explicit consideration of the systems involved—neither their interface details nor performance requirements."

Section 4.1.6 of the Guide defines the process of addressing requirements and solution options as both (1) prioritizing and selecting requirements to be implemented and (2) evaluating and selecting technical approaches for meeting those requirements. The following best practices are identified:

- "Working with the constituent systems to identify and assess alternative approaches" and "assessing options for changes in their systems to address the

[SoS] needs" using a "value driven design process to weigh the alternatives in terms of their comparative values to various users";
- Looking "broadly at the set of longer-term needs" and addressing requirements "in ways that practically leverage ongoing system activities";
- Assessing "the full range of issues—to include life-cycle cost, technical and integration risk, etc." through trades when making "decisions about which systems changes should be made in an increment of SoS development";
- Identifying conflicts and assessing "ways to mitigate the risks inherent in them" when "the needs of the systems users conflict with the objectives of the SoS";
- Remaining "aware of the requirements of the systems as well as plans for funding and scheduling changes" to "anticipate impacts of system changes on the SoS";
- Convincing "the systems engineer for a constituent system that it is in the constituent system's interest to change its implementation to meet the SoS needs."

5.1.6 COMPONENT-BUS-SYSTEM-PROPERTY

The CBSP (Component-Bus-System-Property) approach helps to refine a set of requirements by applying a taxonomy of architectural dimensions (Grünbacher, Egyed et al. 2004). The intent is to provide a generic approach that primarily works with arbitrary informal or semi-formal requirements representations as well as different architecture modeling approaches. Although requirements may also be captured in a formal language (e.g., KAOS), informal or semi-formal approaches are still used very frequently. In particular, CBSP has been integrated with the WinWin requirements negotiation approach, which supports multi-stakeholder elicitation of requirements and captures requirements informally but in a structured fashion.

CBSP provides an intermediate model between the requirements and the architecture that helps to iteratively evolve the two models. For example, a set of incomplete and quite general requirements captured as statements in a natural language might be available. The intermediate CBSP model then captures architectural decisions as an incomplete "proto-architecture" that prescribes further architectural development. The intermediate model still "looks" like requirements but "sounds" like an architecture. The CBSP approach also guides the selection of a suitable architectural style (e.g., client-server, peer-to-peer, layered, dataflow, etc.) to be used as a basis for converting the proto-architectures into an actual implementation of a software system architecture.

Figure 10 shows the CBSP model in the context of the Twin Peaks model suggested in literature for relating requirements and architecture. The Twin Peaks model suggests that requirements and architectures are evolved iteratively and concurrently. In such a context, the intermediate CBSP model can be used at different levels of detail in the modeling process. For example, it can help to refine high-level, informal requirements early in a project and more elaborated requirements in later iterations; or it can also

help to understand how issues arising in architecture modeling and simulation relate to the requirements.

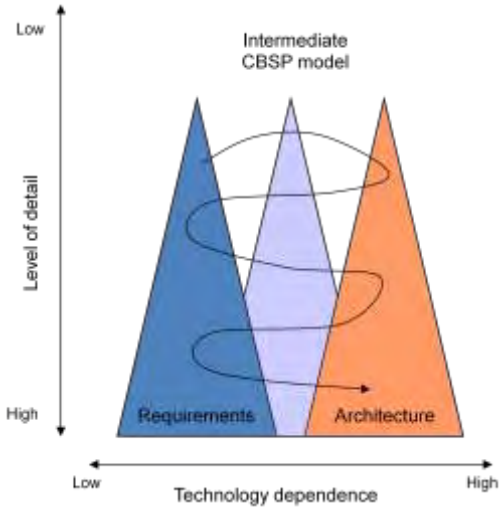

Figure 10: The CBSP in the context of the "Twin Peaks" software development process

CBSP provides:

- a lightweight way of refining requirements using a small, extensible set of key architectural concepts;
- mechanisms for "pruning" the number of relevant requirements, rendering the technique scalable by focusing on the architecturally most relevant set of artifacts;
- involvement of key system stakeholders, allowing nontechnical personnel (e.g., customers, managers, even users) to see the impact of requirements on architectural decisions if desired;
- and adjustable voting mechanisms to resolve conflicts and different perceptions among architects.

Together, these benefits afford a high degree of control over refining large-scale system requirements into architectures, via a five-step process:

1. *Selection of requirements for next iteration* — based on importance to project success and feasibility with respect to technical, economic, organizational, or political constraints on implementing a requirement.
2. *Architectural classification of requirements* — each requirement is rated by the stakeholders for its relevance to the system's **C**omponents, **B**uses (i.e., connectors enabling component interaction), the entire **S**ystem, or system **P**roperties.
3. *Identification and resolution of classification mismatches* — any inconsistencies in how the stakeholders perceive individual requirements' relevance to system architecture must be discussed and resolved.

4. *Architectural refinement of requirements* — each requirement is restated into multiple C, B, S, and/or P statements, based on the requirement's relevance to those dimensions.
5. *Trade-off choices of architectural elements and styles with CBSP* — multiple styles may be possible for a given problem, and the system architects must select the one that will maximize the system's utility to the stakeholders, while minimizing any issues introduced by the selected solution.

5.1.7 COSOSIMO

The Constructive Systems Engineering Cost Model (COSYSMO) is a calibrated cost model we previously developed that most closely estimates the systems engineering effort associated with complex systems (Valerdi 2005). However, COSYSMO only allowed the user to characterize the system using a single set of parameters, with no ability to generate multiple characterizations for the various subsystems comprising a complex system and provided limited capabilities for modeling an SoS. We subsequently introduced a novel technique, Constructive System of Systems Integration Cost Model (COSOSIMO), that estimates the effort associated with the Lead System Integrator (LSI) activities to define the SoS architecture, identifies sources to either supply or develop the required SoS component systems, and eventually integrates and tests these high level component systems (Lane and Boehm 2008).

For the purposes of COSOSIMO estimation, an SoS is defined as an evolutionary net-centric architecture that allows geographically distributed component systems to exchange information and perform tasks within the framework that they are not capable of performing on their own outside of the framework. The component systems may operate within the SoS framework as well as outside of the framework, and may dynamically come and go as needed or available. Based on the feedback obtained from different industrial partners, COSOSIMO was designed with three constituent sub-models: a planning/requirements management/architecture (PRA) sub-model, a source selection and supplier oversight (SS) sub-model, and an SoS integration and testing (I&T) sub-model. Next, we briefly summarize these sub-models.

The LSI PRA activities are those associated with SoS concept development; requirements identification, analysis, and evolution; SoS architecture development and evolution, as well as the long term planning for providing incremental SoS capabilities in accordance with the SoS sponsor's cost and schedule targets.

The LSI SS activities are those associated with the identification of potential component system suppliers or vendors, the development of Requests for Proposals (RFPs) and statements of work for candidate suppliers/vendors, the evaluation of supplier/vendor responses, the selection of suppliers/vendors, and then the on-going oversight of supplier/vendor performance through delivery and validation/verification of the desired component system.

The LSI I&T activities are those associated with the SoS component system integration and the verification/validation testing at the SoS level. These activities include integration and test planning, set up of the integration and test environments and tools, development of test data and procedures, and the actual execution and tracking of integration and verification/validation tests.

The COSOSIMO parameters include of the number and complexity of both SoS and constituent system requirements, the complexity of interface protocols between constituent systems, team-related parameters (e.g., cohesion and capabilities), the estimates of the architecture, process and tool maturity, the compatibility of cost/schedule in the SoS and constituent systems, numbers of operational scenarios, as well as additional parameters in relation to the software suppliers.

5.1.8 PROCESS SIMULATION

Process simulation is a relatively mature tool for assessing the effectiveness of different processes with respect to cost and schedule, while incorporating the effect of uncertainty. Typically, simulation is used in an experimental mode to assess the effect of different combinations of independent variables, or in a what-if analysis mode to determine the effect of changes in baseline system. In the case of integration, simulation models could be used to determine the effectiveness of different decision prioritizations.

Two primary methods for process simulations are discrete-event simulation (Law and Kelton 2000) and system dynamics simulation (Sterman 2000). Discrete-event simulation models processes or systems at the transactional level, representing the different events/transactions that can occur, changing the state of the process/system and scheduling events to occur in the future. Historical uses of discrete-event simulation include factory and supply chain performance. Discrete-event models have been used to analyze DoD acquisition programs, specifically the effectiveness of evolutionary acquisition, system modularity and production level on cost (Bodner, Rahman et al. 2010). One such process model is shown in Figure 11.

System dynamics represents a process or system as a continuous set of accumulation variables (e.g., cost or performance) that are affected by non-linear phenomena such as feedback and lags. It also has been used to study acquisition. Madachy (2008) describes, in particular, acquisition of software-intensive systems, building on the work of Abdel-Hamid and Madnick (1991). Other acquisition-related work is documented in Ford and Dillard (2009).

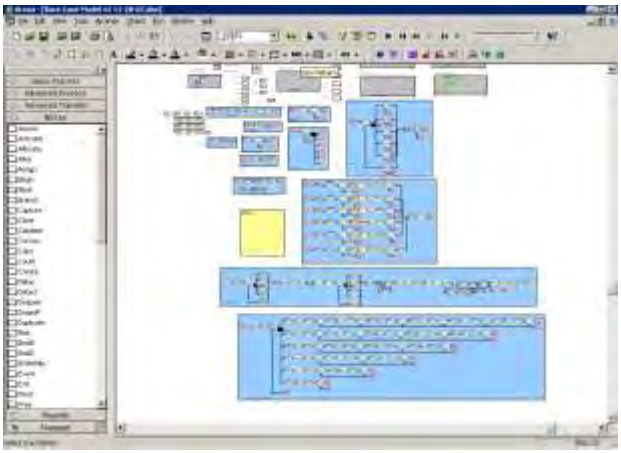

Figure 11: Example discrete-event acquisition model

One limitation of process simulation is that it does not capture the motivations and interactions of various actors in a net-centric enterprise. One promising technology in that regard is agent-based simulation (Hillebrand and Stender 1994). Agent-based simulation models actors and their interactions and is increasingly used in social science applications. Agent-based simulation is relatively immature and requires extensive programming to represent realistic actor behaviors. A reference model to support organizational modeling and simulation, combining these different simulation paradigms, is described in (Bodner 2009).

Little, if any, work has used simulation to study the process of IT system merger/integration.

5.2 Integrated Methodology

This section presents the methodology specified by this phase of the research.

5.2.1 Overall methodology

The overall methodology is shown in Figure 12. The left side of the figure represents capabilities articulated by the set of net-centric actors that are collaborating in a venture that will involve some type of joining of IT systems. These capabilities are decomposed into requirements and architectures. As this process unfolds, the white arrow indicates the ability to provide traceability on the progress of achieving capabilities. Of course, the needs and missions of the net-centric actors evolve over time, as do the desired capabilities. This has implications, of course, for requirements and architectures.

The right side of the figure illustrates the decision process used to evolve capabilities into requirements and architectures. First, a reconciliation process is used to identify conflicts between capabilities or requirements of different actors in the net-centric enterprise and then to negotiate settlements. Then various decision drivers are identified and categorized according to context variables and constraint variables

identified earlier in the decision framework. Once this is done, a decision process is invoked to prioritize decisions according to a framework that incorporates value and risk to the enterprise.

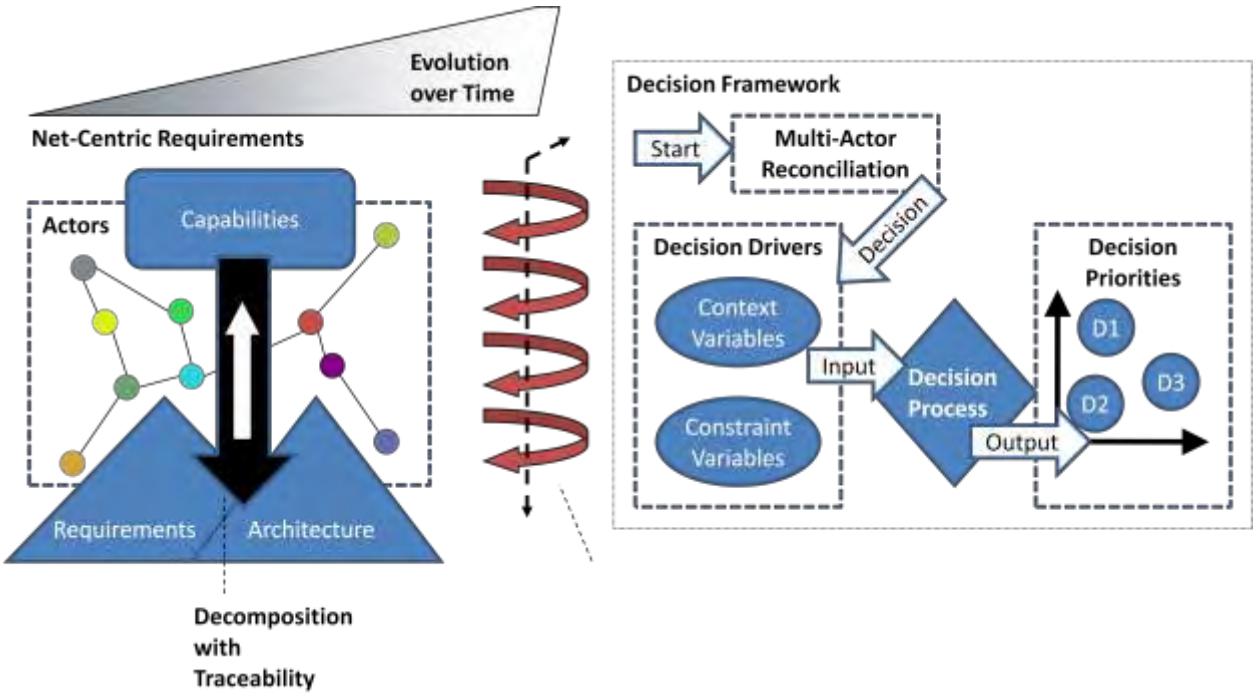

Figure 12: Overall methodology

The red arrows indicate that this is a spiral process involving iteration between the decision process and a general movement down from capabilities to requirements and architectures. Note that this spiral process may revisit decisions if the net-centric evolution makes this necessary.

5.2.2 Decision Inputs

The first step in the decision framework is to determine the system I&M decision drivers. Decision drivers are characteristics of the intended integrated/merged system or SoS that have a critical impact on the value, cost, or risk associated with a decision option. Recall that decision drivers can be roughly categorized as either context variables (explained in Section 4.3) or constraint variables (explained in Section 4.4).

In the proposed methodology, decision drivers are derived from the capabilities, requirements, and architecture of the system. Thus, decision drivers exist at all levels of the system hierarchy/decomposition, and can be used to inform decision-making at all levels. Furthermore, the use of the methodology for decision-making may alter the system capabilities, requirements, or architecture, which in turn alters the decision drivers. For this reason, the methodology should be used in an iterative fashion until

the capabilities, requirements, and architecture are stable (the definition of "stable" and the process for determining stability are the subject of future research).

Figure 13 illustrates the process of determining I&M decision drivers. As shown, the process may take place at the level of capabilities and the level of requirements and architecture. However, it is not yet known what the consequences may result from performing this process as multiple levels simultaneously. Therefore, the methodology currently recommends iterating only at one level at a time.

As shown in Figure 13, the methodology requires capabilities to be linked to requirements and architecture through refinement (in the downward direction) and traceability (in the upward direction). Refinement and traceability are needed because decisions made using the methodology that affect the system capabilities will also affect the system architecture, and vice-versa. There are existing MPTs, with varying levels of maturity, for achieving refinement and traceability, particularly between requirements and architectures. MPTs for refinement and traceability between capabilities and requirements are not currently sufficiently mature for use in production, large-scale engineering, and are the subject of future research. For refinement and traceability at the requirements/architecture level, CBSP (described in Section 5.1.6) is a strong candidate MPT.

In the context of the net-centric ecosystem, capabilities, requirements and architecture should not be dictated by a central authority – doing so results in suboptimal I&M decisions. Instead, capabilities, requirements and architecture should be determined through negotiation among all system stakeholders. Where conflicts exist among stakeholder goals and expectations, negotiation must reconcile these conflicts. Multiple MPTs have been proposed for performing negotiations among parties with different and conflicting goals. The WikiWinWin approach is a best-of-breed MPT for reconciling issues among multiple stakeholders.

Some decision drivers are more impactful at a particular level of the system decomposition. For example, as shown in Figure 13, the integration type is central to architectural decision-making, but may be less significant to capability decision-making. On the other hand, information access is critical to capability decision-making, but is potentially less impactful to architectural decision-making. The categorization of decision drivers shown in Figure 13 is a rough one, and is not intended to imply a strict division.

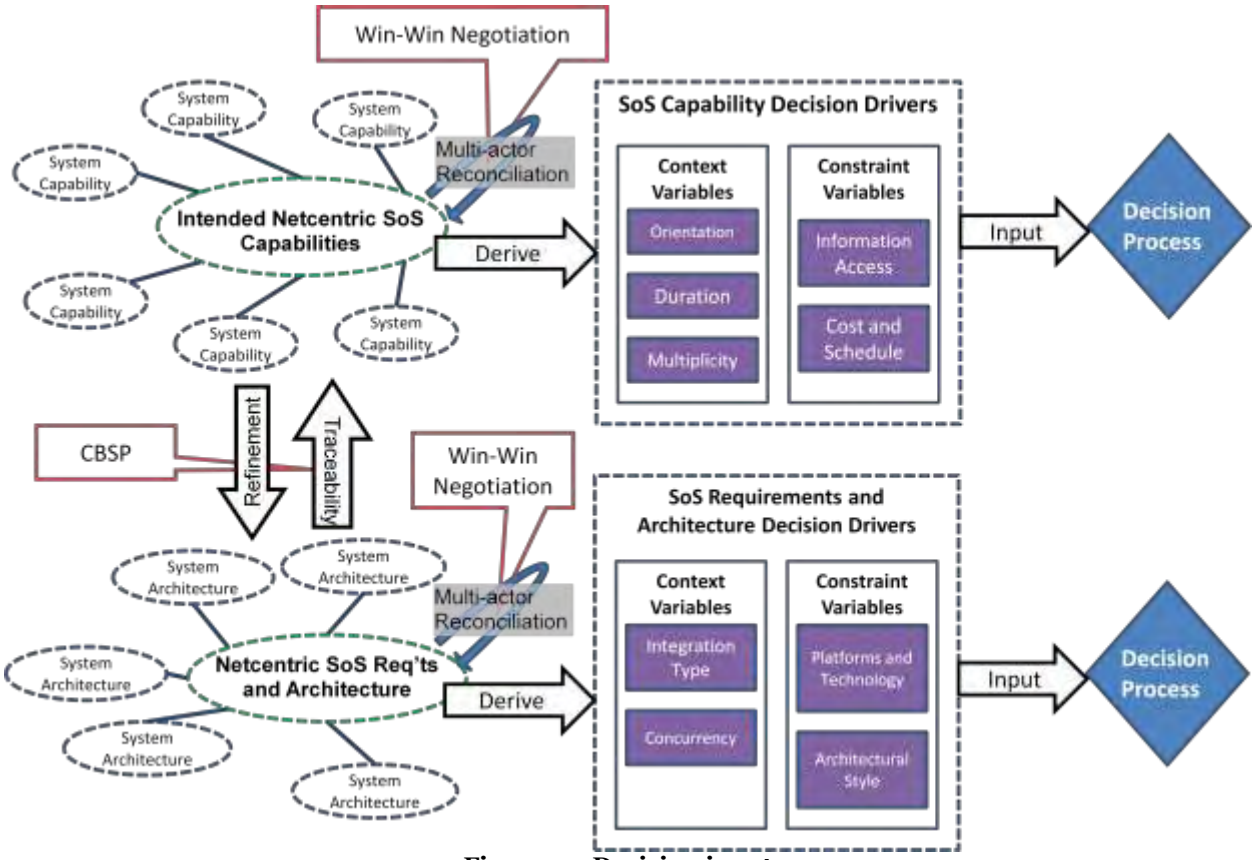

Figure 13: Decision inputs

5.2.3 Decision Process

The second step in the decision framework is to use the indentified decision drivers to characterize the value, cost, and risk of decision options. The value and cost assigned to decision options does not need to be an exact numerical quantification; rather, these quantities can be specified in relative, qualitative terms.

Figure 14 illustrates the process used to characterize the value, cost, and risk of decisions. For the purposes of the methodology, cost and risk are combined into a single value termed complexity. Figure 14 shows the process as it takes place at the level of capabilities; however, note that this process may also take place at the level of requirements and architecture.

As shown in Figure 14, two paths are taken through the decision process for each intended net-centric capability. The first path characterizes the value of the capability, while the second path characterizes the complexity of realizing the capability. Characterizing the value of capabilities is a non-trivial problem. The DoD Systems

Engineering Guide for Systems-of-Systems, Section 4.1.1, provides some guidance as to how this can be accomplished. However, further work in this area is needed.

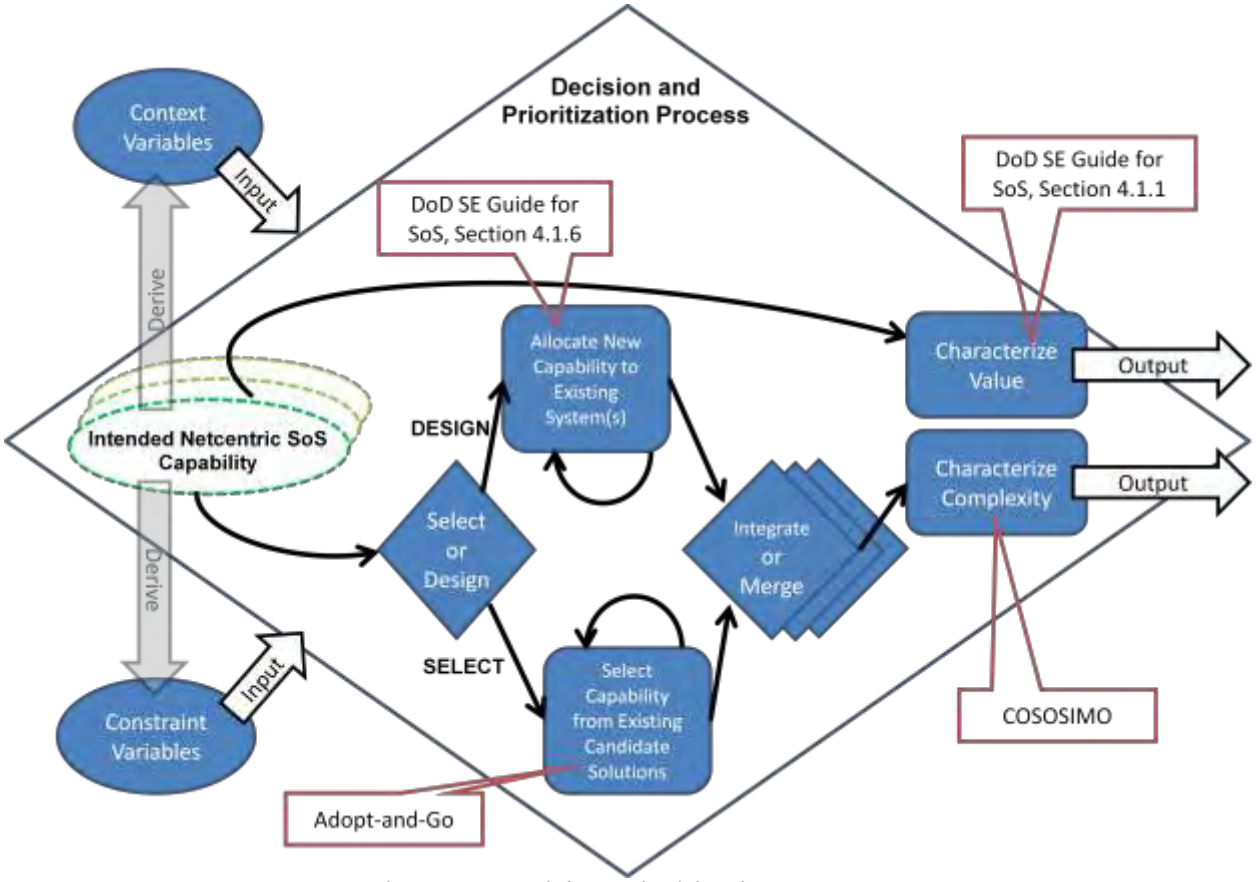

Figure 14: Decision prioritization process

Characterizing the complexity of realizing a capability is a somewhat better-understood problem. Since the proposed methodology is intended to be used in the context of mergers and integrations, the first step is to determine, for each capability, whether the capability can be selected from an existing subsystem, or whether the capability must be designed (that is, implemented through new development or the aggregation of subsystem capabilities). If one or more subsystems already implement the capability, then the capability can be selected. This is common when similar systems are being merged (for example, two document management systems). Adopt-and-Go is one good MPT for selecting capabilities from existing candidate solutions or implementations. However, if no subsystem already implements the capability, the capability must be designed. In the I&M context, the design of a capability usually does not mean designing from a "blank slate." Rather, design involves the aggregating, extending, and improving the capabilities of subsystems; the DoD SE Guide for SoS, Section 4.1.6, outlines best practices.

Once the capability has either been selected or designed, the complexity of realizing it is characterized. This part of the process uses the various decision drivers as input to a complexity model, such as COSOSIMO.

5.2.4 DECISION PRIORITIES

The third and final step in the decision framework is to use the value and complexity characterizations to arrive at I&M decisions and correspondingly refine, extend, or otherwise modify the system capabilities, requirements, and architecture. Since each capability, requirement, or architectural decision that is analyzed through the process described above is assigned a value and a complexity, each decision can be placed within the two-dimensional prioritization plane (as discussed in Section 4.5). Recall that, in this plane, the x-axis represents complexity, while the y-axis represents value. Decisions that reside in the upper-left quadrant of the plane are "low-hanging fruit."

There are numerous types of I&M decisions that may be informed through the outputs of the decision process outlined above. Figure 15 depicts four categories of decisions (although this is not an exhaustive list):

- *Strategy decisions* refer to the overall approach used to achieve an integration or merger.
- *Process decisions* refer to the design, development or implementation methods used.
- *Mechanism decisions* refer to the specific technical solutions pursued.
- *Resource allocation decisions* refer to the assignment of specific tasks and jobs to people and teams.

As decisions are made, the system or SoS capabilities, requirements, and architecture may be updated.

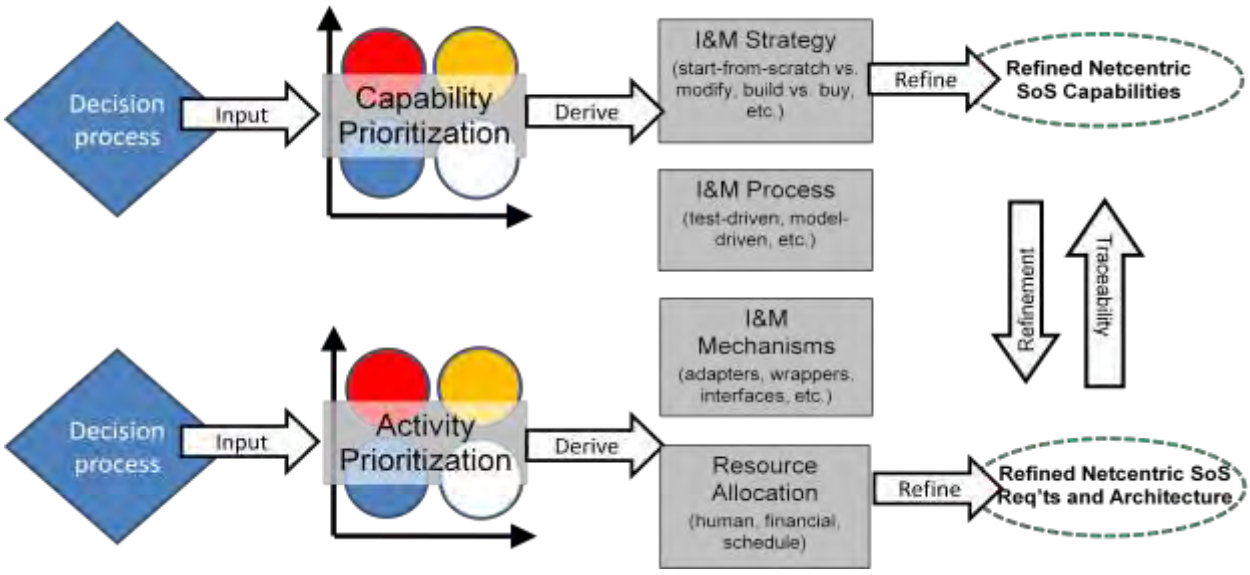

Figure 15: Decision priorities

5.2.5 PROCESS OF USE

An algorithmic description of using the methodology is shown below.

- Initialization
  - Articulate top-level capability intents
  - Identify net-centric actors at all levels
  - Identify existing systems and architectures
  - Reconcile capability intents into capabilities
- Iteration in decision framework
  - Decompose top-level capabilities into functions into requirements
  - Reconcile capabilities, functions, requirements and architectures
  - Map capabilities/functions/requirements to architectures (design vs. select, integrate vs. merge)
  - Incorporate evolving needs over time
- Progress reporting
  - Traceability on progress of top-level capabilities
  - Volatility and conflict assessment
  - Re-prioritization, re-budgeting, re-scheduling

5.3 Topics for Further Research

To realize the methodology in a more mature form, a number of topics have been identified that require further research.

- Representation framework for capabilities, functions and requirements
- Design vs. select guidelines for mapping of requirements to architectures
- Compatibility assessment between architectures (reconciliation of architectures/styles)
- Progress metrics for capabilities and algorithms for traceability
- Decision support for activity prioritization
    - Decision categories (capabilities, mechanisms, resources)
    - Outcome metrics (cost, risk, schedule)
- Net-centric reconciliation support

6 Conclusion and Future Research Directions

This report has documented the results of a Phase 1 research project investigating methodologies and MPTs to support requirements management in a net-centric context. Six case studies are analyzed within a framework that serves to elicit important considerations for managing requirements during system integrations and mergers.

The case studies range from business mergers, to inter-agency and inter-service government systems, to public-private IT systems that operate under a highly regulated environment. The framework is used to specify the type of integration or merger, the context in which the integration or merger takes place (IT type, vertical or horizontal orientation, duration of integration/merge, number of systems involved and concurrency), the constraints involved (platforms/technologies, architectural style, information access, cost and schedule and external constraints). These elements affect the risk and value of decisions related to the integration/merging, allowing prioritization.

A number of potentially useful MPTs are identified and discussed that address specific aspects of the overall problem. A methodology is then proposed to address requirements management problem that incorporates the MPTs and the decision framework that facilitates prioritization of activities.

Future work involves further refinement and elaboration of the methodology, enhancement of the various MPTs to fit better with the net-centric context, further case

study analysis to support the methodology development, and validation of the methodology and MPTs using a real system. In addition, several supporting research topics have been identified that would facilitate methodology development. High priority topics for future research include (i) representation framework for capabilities, functions and requirements, (ii) design vs. select guidelines for mapping of requirements to architectures and (iii) decision support for activity prioritization.

APPENDICES

APPENDIX A: CASE STUDY QUESTIONNAIRE AND DATA SPECIFICATION

As part of the research investigating case studies, the following questionnaire and data specification document was created. The front matter summary is to be presented to the principals of the case study to promote their understanding of the research and underlying rationale and approach. The questions posed may be useful in general analysis of IT merger and integration situations.

A.1 FRONT MATTER SUMMARY

The goal of this case study is to support a research project addressing requirements management in a net-centric environment. Here, the term net-centric environment means an enterprise with multiple organizations under a common umbrella that operate semi-autonomously. These organizations often have joint IT system needs, and they experience these needs in a constantly evolving external environment of opportunities and threats.

The fundamental question is how to manage requirements for these IT systems, given the multi-stakeholder, hierarchical enterprise and constantly evolving environment. At the enterprise level, top-level capabilities are specified. These are decomposed into intermediate functions at various levels in the enterprise hierarchy. At the lowest level, requirements are specified for software developers. These capabilities, functions and requirements must then be mapped to software architectures. Of course, legacy systems strongly influence what can be done in terms of architecture selection and design. Finally, these processes operate under significant time pressure, typically resulting in ad-hoc decisions.

We can make the following observations, which serve as avenues for research.

- With the multi-organizational nature of the enterprise, each organization with its own mission and operating model, there are likely to be conflicting capabilities, functions and requirements. How best can these be identified and reconciled?
- During the requirements specification and management process, there is typically not a single perspective or knowledge source for progress and status, given the hierarchical and multi-organization nature of the enterprise. Thus, how does an executive-level decision-maker know the "progress" for development and deployment of a capability?
- In the decision process whereby capabilities are translated into IT systems, there are different decision points and alternatives. Constraints may be externally

imposed or may derive from systems/technologies. Needs evolve over time. What types of decision tools and support can be provided to aid decision-makers in understanding the implications of their decisions on cost, schedule and risk?

This research project is investigating these issues in the context of corporate mergers and acquisitions in an effort to understand issues involved in a multi-organization enterprise. This document proposes interview questions and other data elements that would be support research goals. Several different topical areas are noted, then within each area questions and data elements are specified.

A.2 Intent and Actors

- Who were the actors in the integration?
  - Independent units,
  - Different departments under the same organizational unit, or
  - Units from different organizations?
- What was the intended duration of the integration?
  - Temporary and of known duration.
  - Temporary and of unknown duration.
  - Permanent and known to be permanent.
- What were the plans to unwind the integration (contingency plans if the intent was to be permanent)?
- Was the intent to integrate the actors' existing capabilities or to create new, emergent capabilities?
- What types of information did the actors share between each other? For example, did they share interfaces, metadata, high-level capabilities, internal requirements repositories, access to the internal systems, etc.?
- How was the integration strategy defined for each actor? Did they create a mutual list of capabilities or each actor had its own set of capabilities and requirements with respect to the integration?
- How were the requirements and architectural conflicts communicated between the different actors?
- How did the separate systems evolve during the integration? Did their respective requirements change in any significant way that required changes to the integration itself?
- If you had to do this again, what teaming structures and management would you like to have?

A.3 Decision-Making

- How did the process of determining/eliciting capabilities work?
- How did you arrive at an overall integration strategy?
- At what level of abstraction was the integration strategy defined? How was it translated to lower-level integration tasks?

- How did you determine how to allocate human and financial resources?
- Did you prioritize certain integration tasks over others? If so, how did you determine the prioritization?
- How did you keep track of the overall integration progress or the progress on a key capability?
    - How would you ideally measure the integration progress?
- What would you ideally see in support of decision making (e.g., prioritization of new capabilities, quantitative cost estimates, a list of possible obstacles or architectural conflicts)?
- What types, if any, of architectural integration conflicts occurred? How were they resolved? Were there any particularly useful architectural styles that enabled seamless integration?
- Were any established methods, processes or tools (MPTs) used to support the integration process? If so, to what extent were they used in conjunction with improvised approaches needed to meet deadlines?

A.4 Integration Context

- Which of the following types of integration were attempted?
    - User interface integration – providing a common user interface to multiple systems
    - Data integration – creating common data repositories and formats for multiple systems
    - Control integration – enabling multiple systems to directly invoke each other's services and functions
    - Process integration – utilizing multiple systems seamlessly within business processes
- Would you characterize the integration as
    - Primarily horizontal (i.e., integrating systems that implement similar capabilities),
    - Primarily vertical (i.e., integrating systems with different functions to obtain a higher-level capability),
    - Neither, or both?
- Was the integration intended to be permanent, flexible or temporary?
- How many distinct systems needed to be integrated?
- How dynamic was the integration? Were there other integrations involving the same systems started or ongoing in parallel?

A.5 Integration Constraints

- What constraints were imposed externally? Were these all known at the beginning, or were any imposed mid-process?

- How did the existing platforms and technologies (e.g., operating system, middleware, and programming languages) affect the integration process and integration decisions?
- Did the architectural styles and patterns in the systems targeted for integration affect the integration process and integration decisions? If so, how? Please include both positive and negative impacts.
- Was sufficient technical documentation for all involved systems available?
    - Was any technical information unavailable due to security, intellectual property, or other concerns?
- Did cost and schedule constraints play an important role in prioritizing integration activities?
- What kind of a role did additional organizational constraints play (e.g., integrating systems that were used by multiple departments, security/need-to-know across departments, levels of access by department)?
- Were there constraints regarding availability of the constituent systems to their users while the systems were under integration?
- Was there system downtime planned to support migration to system upgrades? If so, how frequent, extensive were these?
- Were there requirements regarding the reversibility of the integration?

A.6 Capabilities and Requirements

- What processes, methods, and tools were used to identify and reconcile conflicts in capabilities, functions, requirements and constraints from different sources in the requirements development and decomposition process?
- To what extent were rationales for capabilities, functions or requirements captured during this process?
- How did the process of decomposing capabilities into requirements work? Was there iteration? Were there generic ways of representing the different functions in the decomposition process? For example, was there a standard number of levels in the composition?
- What were the specific function and requirement artifacts that were developed, as well relationships among them and between them and capabilities and constraints?
- To what extent were there changes in capabilities or constraints during the integration process, and how did those changes impact the functions/requirement decomposition process? What were specific changes and their timing?
    - Internally imposed changes
    - External imposed changes
- Did a capability ever get removed due to determining it was discovered too difficult to achieve given time and budget? Was a proposed capability change ever rejected on similar grounds?

A.7 ARCHITECTURES

- What architectures existed at different levels, ranging from the enterprise level to the department level?
- What legacy issues significantly impacted the integration process?
- To what extent were existing architectures and systems selected versus designed in the integration process? Can you describe how this was done and various rationales for select vs. design?
- What types, if any, of models were used in this process?
- What elements do you typically map requirements to?
- What constraints played into the process of mapping requirements to architectures?
- Were there trade-off tools that factored in cost and schedule effects/risks on architectural decisions?
- To what extent did the decomposition of capabilities to requirements overlap with the mapping of capabilities/requirements to architectures?
- Were there changes in capabilities, constraints, or requirements during the integration process?
- If so, what was the extent/number of changes? Rate of change?
- If so, how did such changes impact the architecture mapping process?
- To what extent were there conflicts:
    - Between architectures from different systems that were being integrated, or
    - Between capabilities/functions/requirements on the one hand and architectural possibilities on the other, given legacy constraints?

A.8 PROBLEMS AND EXCEPTIONS ENCOUNTERED

- What problems and exceptions occurred in the integration process not captured in the above questions and data elements?
- What should have happened in the integration process that would have improved things, as opposed to what did happen? What were factors that caused this divergence?
- If you had to do this again, what types of methods, processes, tools and artifacts would you like to have ideally? How should the integration process have worked?

APPENDIX B: REFERENCES

Abdel-Hamid, T. and S. Madnick (1991). Software Project Dynamics: An Integrated Approach. Englewood Cliffs, NJ, Prentice Hall.

ANSI/IEEE (2006). "ANSI/IEEE Standard 1471: Recommended Practice for Architectural Description of Software-Intensive Systems." Retrieved April 1, 2011, from http://www.iso-architecture.org/ieee-1471/.

Baldwin, C. and D. Lane (2003). Compaq's Struggle. Boston, MA, Harvard Business School, Harvard College.

Basole, R. C. and R. A. DeMillo (2006). Enterprise IT and Transformation. Enterprise Transformation: Understanding and Enabling Fundamental Change. W. B. Rouse. Hoboken, NJ, Wiley-Interscience**:** 223-252.

Beer, M., R. Khurana, et al. (2005). Hewlett-Packard: Culture in Changing Times. Boston, MA, Harvard Business School, Harvard College.

Bell DeTienne, K. and C. L. Hoopes (2004). "The Hewlett-Packard and Compaq Merger: A Case Study in Business Communication." Education Review of Business Communication **1**(1): 27-46.

Bodner, D. A. (2009). A First-Generation Reference Model for Organizations to Support Organizational Simulation. Atlanta, GA, Tennenbaum Institute, Georgia Institute of Technology.

Bodner, D. A., F. Rahman, et al. (2010). Addressing Cost Increases in Evolutionary Acquisition. Proceedings of the 2010 Acquisition Research Symposium. Monterey, CA, Naval Postgraduate School. **1:** 329-345.

Boehm, B., P. Grunbacher, et al. (2001). "Developing Groupware for Requirements Negotiation: Lessons Learned." IEEE Software: 46-55.

Boehm, B. and A. Jain (2005). An Initial Theory of Value-Based Software Engineering. Value-Based Software Engineering. S. Biffl, A. Aurum, B. Boehm, H. Erdogmus and P. Grunbacher, Springer-Verlag**:** 15-38.

Burgelman, R. A. and P. E. Meza (2004). HP and Compaq Combined: In Search of Scale and Scope. Stanford, CA, Graduate School of Business, Stanford University.

CJCS (2007). Operation of the Joint Capabilities Integration and Development System. Washington, DC, Chairman of the Joint Chiefs of Staff, Department of Defense.

DoD (2008). Systems Engineering Guide for Systems-of-Systems, Version 1.0. Washington, DC, OUSD(A&T)SSE.

Ford, D. N. and J. Dillard (2009). "Modeling the Performance and Risks of Evolutionary Acquisition." Defense Acquisition Review Journal: 143-158.

Goldstein, H. (2005). "Who Killed the Virtual Case File?" IEEE Spectrum **42**(9): 24-35.

Grünbacher, P., A. Egyed, et al. (2004). "Reconciling Software Requirements and Architectures with Intermediate Models." Software and Systems Modeling **3**(3): 235-253.

Hillebrand, E. and J. Stender (1994). Many-Agent Simulation and Artificial Life. Amsterdam, IOS Press.

Laguna, M. and J. Marklund (2004). Business Process Modeling, Simulation, and Design, Pearson/Prentice Hall.

Land, R. and I. Crnkovic (2011). "Oh Dear We Bought Our Competitor: Integrating Similar Software Systems." IEEE Software **28**(2): 75-82.

Lane, J. A. and B. Boehm (2008). "System of Systems Lead Systems Integrators: Where Do They Spend Their Time and What Makes Them More or Less Efficient." Systems Engineering **11**(1): 81-91.

Lane, J. A. and T. Bohn (2010). Using Models to Understand and Evolve SoSs. Proceedings of the 2010 International Congress on Ultra Modern Telecommunications and Control Systems.

Law, A. M. and D. W. Kelton (2000). Simulation Modeling and Analysis. New York, McGraw-Hill.

Madachy, R. (2008). Software Process Dynamics. Washington, DC, Wiley-IEEE Press.

Mark, K. and J. Mitchell (2004). Hewlett-Packard in 2001. London, Ontario, Richard Ivey School of Business, University of Western Ontario.

Mehta, M. and R. Hirschheim (2004). A Framework for Assessing IT Integration Decision-Making in Mergers and Acquisitions. Proceedings of the 37th Annual Hawaii International Conference on System Sciences.

Palepu, K. and J. Barnett (2004). Hewlett-Packard-Compaq: The Merger Decision. Boston, MA, Harvard Business School, Harvard College.

Perlow, L. and L. Kind (2004). The New HP: The Clean Room and Beyond. Boston, MA, Harvard Business School, Harvard College.

Rouse, W. B., Ed. (2006). Enterprise Transformation: Understanding and Enabling Fundamental Change. Hoboken, NJ, Wiley-Interscience.

Sarang, P., M. Juric, et al. (2006). Business Process Execution Language for Web Services, Packt Publishing.

Sterman, J. D. (2000). Business Dynamics: Systems Thinking and Modeling for a Complex World. Boston, McGraw-Hill.

Suri, D. (2009). Analysis of Service Oriented Architecture in a Hospital Emergency Room Environment, Masters thesis, Department of Computer Science, San Diego State University, San Diego, CA.

Taylor, R. N., N. Medvidovic, et al. (2009). Software Architecture: Foundations, Theory and Practice, Wiley.

Valerdi, R. (2005). The Constructive Systems Engineering Cost Model (COSYSMO), Ph.D. dissertation, University of Southern California, Los Angeles, CA.

van Lamsweerde, A. (2001). Goal-Oriented Requirements Engineering - A Guided Tour. Proceedings of the Fifth IEEE International Conference on Requirements Engineering**:** 249-263.

van Lamsweerde, A. and E. Letier (2001). "Handling Obstacles in Goal-Oriented Requirements Engineering." IEEE Transactions on Software Engineering **26**(10): 978-1005.

Weilkiens, T. (2008). Systems Engineering with SysML/UML: Modeling, Analysis, Design, Morgan Kaufmann/The OMG Press.

White, S. A. and D. Miers (2008). BPMN Modeling and Reference Guide, Future Strategies.

Wu, D., D. Yang, et al. (2009). Finding Success in Rapid Collaborative Requirements Negotiatioon Using Wiki and Shaper, Technical report TR 2009-519, Center for Systems and Software Engineering, University of Southern California.